Universidad Carlos III de Madrid

DEPARTAMENTO DE SISTEMAS Y AUTOMÁTICA
GRADO EN INGENIERÍA ELECTRONICA INDUSTRIAL Y AUTOMÁTICA

OPTIMIZACION DE UN CONTROLADOR FUZZY MEDIANTE DIFFERENTIAL EVOLUTION

TRABAJO FIN DE GRADO

Autor: José Antonio Hernaiz Navas
Director: Luis Enrique Moreno Llorente
Tutor: Luis Santiago Garrido Bullón

Septiembre 2014

TRABAJO FIN DE GRADO

OPTIMIZACION DE UN CONTROLADOR FUZZY MEDIANTE DIFFERENTIAL EVOLUTION

Autor: José Antonio Hernaiz Navas
Director: Luis Enrique Moreno Llorente
Tutora: Luis Santiago Garrido Bullón

EL TRIBUNAL CALIFICADOR

Presidente:

Secretario:

Vocal:

Una vez realizada la defensa y lectura del Trabajo Fin de Grado en Septiembre de 2014 en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, en Leganés, se acuerda conceder la calificación de:

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos:

Se acaba una etapa intensa que ha durado cinco años de mi vida. Mirando atrás, veo el chico que entró en la universidad con tanta ilusión como temor. Como colofón, ha resultado que había luz al final del túnel y no puedo evitar acordarme de momentos difíciles en los cuales llegué a dudar si sería capaz de terminar. Quiero aprovechar la ocasión para agradecer a mucha gente su apoyo durante este tiempo.

En primer lugar, a mi madre y mi padre, cuyo sacrificio no olvidaré, junto a mi hermana y mi novia por compartir conmigo los momentos de alegría y celebración, pero también soportar con paciencia mis enfados y mal humor en mis adversidades. No han dejado de apoyarme en ningún momento y sé que sin ellos no habría sido posible. También a mis tíos y primos, quienes no han parado de preocuparse por mis progresos todo este tiempo. Asimismo, a mis abuelos, quienes están orgullosos de que haya aprovechado la oportunidad de estudiar, algo que ellos no pudieron hacer.

A mi enorme grupo de amigos, como Mariano, Alberto o Cristian entre muchísimos otros que no me cabe mencionar y que no habéis parado de animarme desde el instituto. Y otros como Dani (mi gran vecino) o Rubén, a los que conozco desde antes de tener uso de razón. A Miguel, mi gran compañero de tantas prácticas y trabajos. Nuestros caminos se separan pero la amistad permanece.

Y a Borja, que fue mi compañero en el colegio y años después nos encontramos en selectividad para decidir cursar la misma carrera. Has sido una pieza fundamental en este logro y quiero agradecerte individualmente todo lo que me has enseñado y aportado durante todos estos años. Eres un ejemplo a seguir.

Por último, agradecer a Luis Enrique Moreno Llorente y Luis Santiago Garrido Bullón la oportunidad de realizar este Trabajo Fin de Grado junto a ellos. Gracias por el apoyo, ayuda y confianza.

"Si el camino se hace duro, los duros haremos camino."

TABLA DE CONTENIDOS

1 Introducción y objetivo

El "control" es un concepto común y ampliamente usado por la mayoría de las personas en su vida cotidiana. Generalmente hace referencia a la interacción producida entre las personas y las máquinas. Un ejemplo claro sería cualquier conductor llevando su propio vehículo por la carretera. El control automático involucra únicamente a las máquinas. Estamos hablando de un campo de estudio muy amplio dentro de la ingeniería: fuerza y movimiento en robótica, o la velocidad y presión en fluidos. Incluso el cuerpo humano cuenta con mecanismos de control automático: la presión sanguínea o el ritmo cardiaco son procesos biológicos equivalentes al control automático. Esta idea surge para liberar al hombre de tareas repetitivas en sistemas complejos.

El concepto de lógica difusa es muy común entre las personas en su vida diaria, ya que tenemos por costumbre utilizar predicados imprecisos a la hora de determinar la cuantificación cuando nos comunicamos. Ejemplos que usamos de estos predicados son: pequeño, largo, muy, menos, etc.

En el presente trabajo trataremos de realizar el control de diferentes tipos de sistemas utilizando reguladores basados en lógica difusa. Este tipo de controlador es conocido como regulador borroso o *fuzzy*. Pretendemos exponer las características fundamentales de la lógica difusa con el fin de comprobar su utilidad para el control de sistemas. **El peso del trabajo consistirá en optimizar las ganancias de entrada y salida que utiliza este tipo de controlador. Para la calibración de estas ganancias emplearemos *Differential Evolution*.** DE es un algoritmo evolutivo derivativo y multipunto. También trataremos de explicar las peculiaridades de este algoritmo.

"No entiendes realmente algo a menos que seas capaz de explicárselo a tu abuela"

Albert Einstein.

Las especificaciones que nos planteamos cumplir en las respuestas de los sistemas son lograr una estabilización (Ts) de los sistemas en un tiempo máximo de 5 segundos, y esperamos no sobrepasar una sobreoscilación del 30% (Mp) en menos de 2'5 segundos (Tp) por motivos de seguridad.

2 Estado del arte

2.1 Controlador Fuzzy: concepto e historia

La lógica difusa es un tipo de lógica que utiliza predicados vagamente cuantificados o también denominados borrosos. Este tipo de predicado es aquel al que se le aplica un cierto grado de imprecisión e incertidumbre en los elementos de un conjunto. Es decir, se verifica una cierta categoría relativa en una determinada magnitud. Ejemplos de estos predicados pueden ser: *joven o viejo, grande o pequeño, largo o corto, más o menos, mucho o poco…* pero, ¿cuánto de grande?, ¿cuánto de corto? o ¿cuánto más? Por otro lado, es posible diferenciar entre el concepto de 'imprecisión' e 'incertidumbre'. La imprecisión está relacionada con el valor estimado de una magnitud o elemento, mientras que la incertidumbre se relaciona con la veracidad o confianza que tengamos sobre este valor mencionado. Para calificar la imprecisión existen diferentes calificativos como vago, general o ambiguo, pero los más convencionales son difuso o borroso. En estos últimos, entendemos que no poseen un límite claro en el conjunto de valores asociados a los objetos a los cuales se refiere. ¿Cuándo dejamos de ser jóvenes y nos hacemos viejos?, ¿Cuándo deja de ser pequeño un elemento para ser grande? o ¿dónde está el límite entre largo y corto? Este tipo de predicados son comúnmente utilizados entre las personas y con ellos queda reflejado la inexactitud con que nos expresamos.

La lógica difusa ("*fuzzy logic*" en inglés) es capaz de adaptarse mejor al mundo real en el que vivimos. Hasta entonces, los sistemas de control presentaban como principal debilidad el depender excesivamente de entradas concretas, sin cubrir los requisitos de continuidad y cobertura.

En la teoría de conjuntos borrosos se asocia una **función de pertenencia** indicando para cada elemento, en qué grado o medida forma parte de ese conjunto borroso. Es decir, mientras que en los conjuntos clásicos (*crisp*), los elementos pertenecen totalmente o no pertenecen al conjunto, en los conjuntos borrosos encontramos grados de pertenencia relacionados con su universo local utilizando valores en el intervalo [0,1]. (Ver Figura 2.1.)

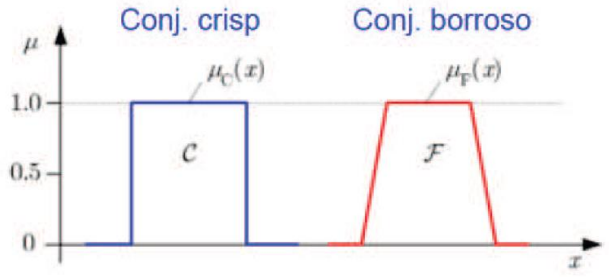
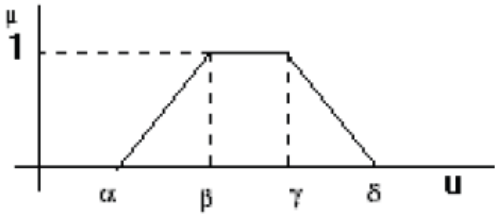


Figura 2.1: conjunto crisp y conjunto difuso o borroso.

La función trapezoidal Π: u → [0, 1] representa proposiciones borrosas, posee cuatro parámetros y se define como:

$$\Pi(u;\alpha,\beta,\gamma,\delta) = \begin{cases} 0 & \text{if } u \le \alpha \\ \frac{(u-\alpha)}{(\beta-\alpha)} & \text{if } \alpha \le u < \beta \\ 1 & \text{if } \beta \le u \le \gamma \\ 1 - \frac{(\gamma-u)}{(\gamma-\delta)} & \text{if } \gamma < u \le \delta \\ 0 & \text{if } u > \delta \end{cases}$$



Por otro lado, 'β' y 'γ' pueden coincidir en valor para conseguir funciones triangulares.

Para conjuntos crisp, las operaciones básicas de unión, intersección y negación se pueden definir sin ambigüedad, pero no ocurre lo mismo en conjuntos borrosos. Zadeh (padre de la lógica borrosa) definió estas operaciones de la siguiente manera:

A) Intersección: El resultado de esta operación será un conjunto borroso formado por aquellos elementos que estén en ambos conjuntos. Desde el punto de vista de la función de transferencia, esta acción sería como la

zona gráfica en la que coincidirían ambos conjuntos en una superposición (Ver Figura2.2):

$$\forall x \in X : \mu A \cap B(x) = \min(\mu A(x), \mu B(x))$$

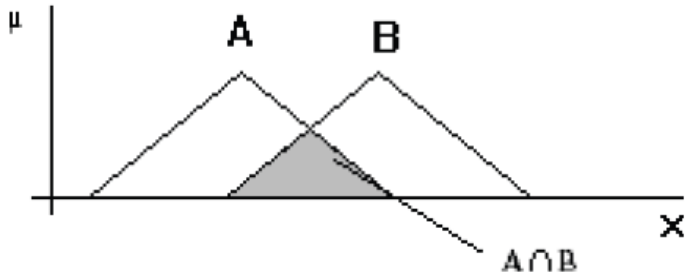


Figura 2.2: Operación de intersección en lógica borrosa.

B) Unión: El resultado es un conjunto en el que se encuentren todos aquellos elementos de un conjunto que no existen en el otro y viceversa. Gráficamente, resultaría ser la superposición de ambas funciones (Ver Figura 2.3):

$$\forall x \in X : \mu A \cup B(x) = \max(\mu A(x), \mu B(x))$$

C)

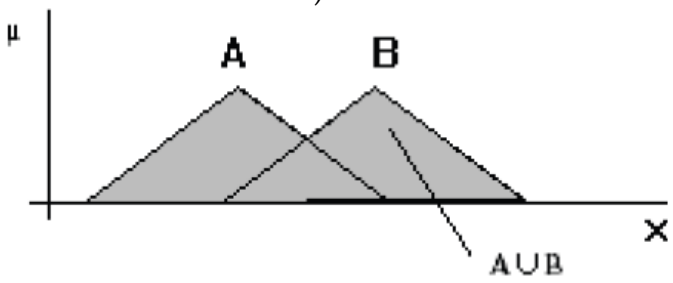


Figura 2.3: Operación de unión en lógica borrosa.

C) Negación: En este caso, nos centramos solo en un subconjunto. El resultado sería un nuevo conjunto con todos aquellos elementos que no formen parte del primer conjunto (Ver Figura 2.4):

$$\forall x \in X : \mu A'(x) = 1 - \mu A(x)$$

D)

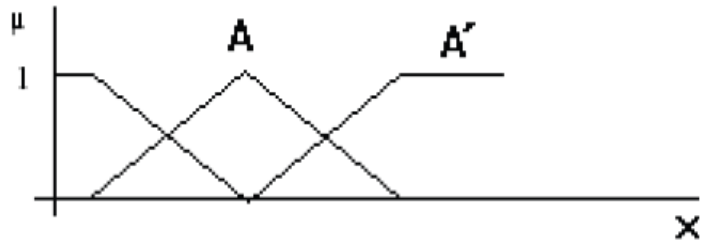


Figura 2.4: Operación de negación en lógica borrosa.

El significado de estas proposiciones borrosas se obtiene por la interpretación de las conectivas '*and*', '*or*' y '*not*', como intersección, unión y negación respectivamente.

Por otro lado, la teoría de conjuntos borrosos cumple además las leyes de Morgan, las propiedades distributiva, conmutativa, asociativa, de monotonicidad e identidad unitaria (o condición de frontera).

Uno de los objetivos de la lógica borrosa es proporcionar las bases del razonamiento aproximado, es decir, utilizar premisas imprecisas para formular respuestas satisfactorias. Esta lógica se basa en reglas heurísticas con la estructura: "SI (*antecedente*) ENTONCES (*consecuente*)", donde tanto antecedente como consecuente son conjuntos difusos. Las relaciones borrosas tiene gran importancia en control, y estas reglas permiten describir las interacciones. Para ello, son imprescindibles otras dos operaciones:

A) Proyección: Convierte una relación ternaria en una binaria, o una binaria en un conjunto borroso, o un conjunto borroso en un valor determinado. (Ver Figura 2.5)

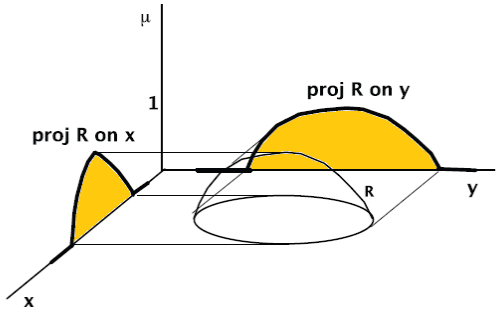


Figura 2.5: Operación de proyección en lógica borrosa.

B) Extensión: Es la operación opuesta a la proyección, esto es, extiende los conjuntos borrosos en relaciones binarias, éstas a ternarias, etc. (Ver Figura 2.6)

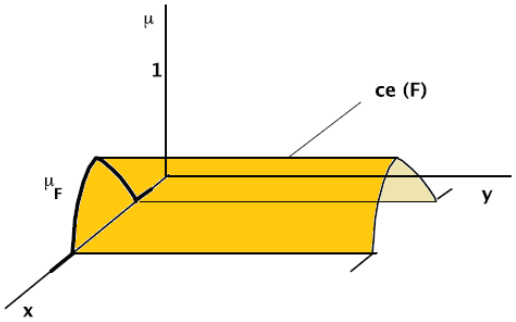


Figura 2.6: Operación de extensión en lógica borrosa.

A continuación en la Figura 2.7 mostraremos un ejemplo visual de cómo se genera la respuesta (*consecuente*), 'R', ante una intersección borrosa:

μA(x), μB(y), cey (A(x)) = ceA(x, y), cex (B(y)) = ceB(x, y) y R = ceA(x, y) ∩ ceB(x, y)

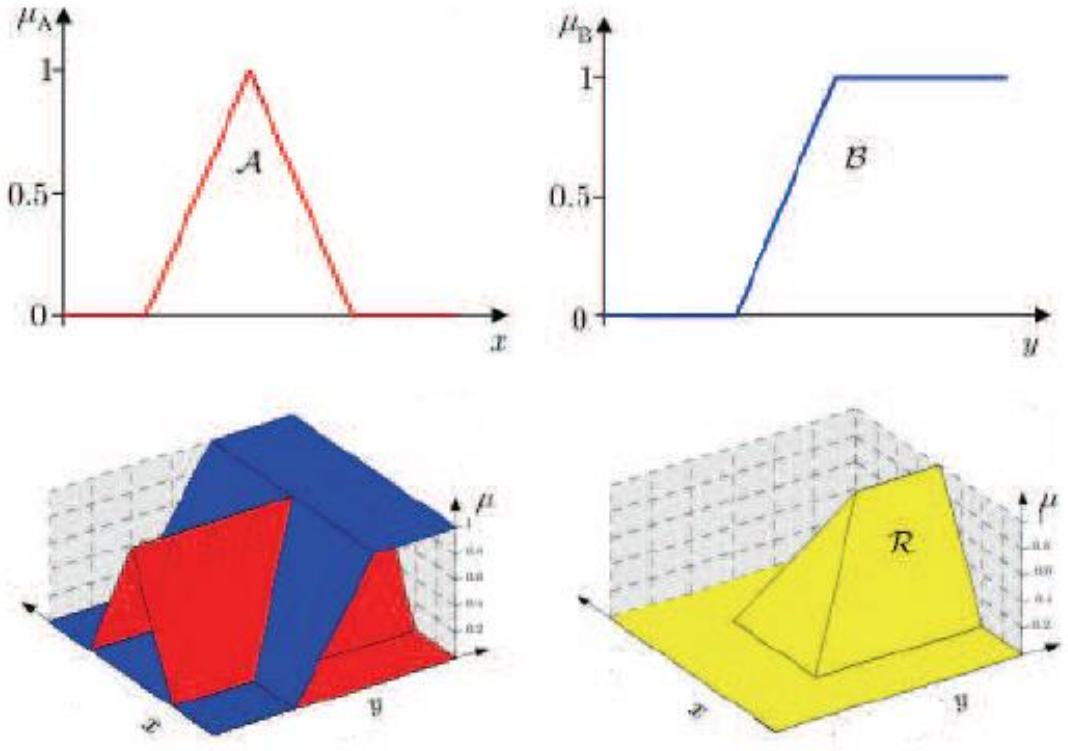


Figura 2.7: Desarrollo de una repuesta borrosa con 2 antecedentes.

Zadeh (padre de la lógica borrosa) también introdujo el concepto de variable lingüística, que se entiende como una variable cuyos valores no son numéricos sino palabras del lenguaje ordinario. Ejemplos de estas variables podrían darse en la altura, la velocidad, la edad, la temperatura o el error absoluto que comete un sistema con respecto a una referencia. En el lenguaje natural solemos utilizar sentencias como: alto o bajito, lento o moderado, joven o anciano, muy frio o poco caliente, desapropiado o ajustado, etc. Resulta imprescindible definir claramente el dominio físico real sobre las variables lingüística. Es conocido como "universo de discurso" y abarca intervalos cuantitativos. Por medio de una función semántica otorgamos un grado, conocido como 'peso', a las variables lingüísticas según donde nos encontremos en nuestro rango de dominio. En la Figura 2.8 podemos observar un ejemplo inventado de cómo podríamos clasificar la madurez de las personas según su edad. El universo de discurso comprendería las edades entre los 0 y los 100 años y utilizaríamos 3 términos lingüísticos con dos etapas de transición entre

ellos. Es muy importante que no se crucen más de dos funciones a la vez para evitar ambigüedades, y deben de hacerlo a la altura de μ=0,5 para una correcta y total definición. En el ejemplo, vemos como los 30 años seria la edad intermedia entre la juventud y la madurez. Podríamos inventarnos más ejemplos con alturas, temperaturas, velocidades, etc.

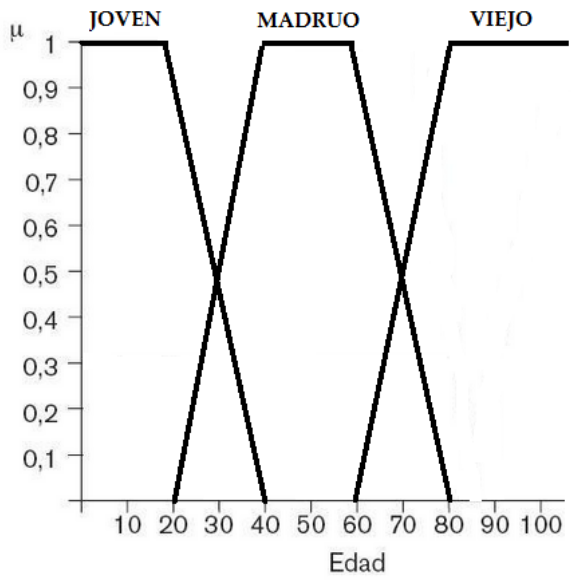


Figura 2.8: Ejemplo de conjunto borroso para la edad de las personas.

Las variables lingüísticas permiten el acercamiento del lenguaje ordinario a la lógica borrosa, facilitando la utilización de este tipo de lógica en el área del control, eludiendo las deficiencias del lenguaje preciso. Esta idea se formalizó en el conocido "*Principio de Incompatibilidad*": A medida que la complejidad de un sistema aumenta, disminuye nuestra capacidad para hacer afirmaciones precisas, incluso significativas, sobre su comportamiento, hasta que se alcanza un umbral más allá del cual precisión y relevancia son características casi mutuamente excluyentes.

Como hemos mencionado anteriormente, la lógica borrosa se basa en premisas imprecisas con la forma estructural "*If-Then*". El operador que relaciona el 'antecedente' y el 'consecuente' es la implicación. La **Implicación Mandani** es una de las más importantes en control borroso y será nuestra elección en el presente trabajo. Su definición se basa en la operación de

intersección, es decir, $p \rightarrow q \equiv (p \wedge q)$, por tanto, el peso de la respuesta será: $\mu Rc\ (x, y) = \min(\mu A(x), \mu B(y))$. Los sistemas tipo Mandani presentan ciertas ventajas como el comportamiento universal, se ajusta con facilidad las entradas y las salidas y se puede utilizar en aplicaciones reales.

En cuanto a la estructura de los sistemas borrosos, se compone de dos elementos fundamentales: la "base de conocimiento" qué esta está formada por el conjunto de reglas que se desarrollan en el diseño del controlador, es decir, contiene la definición de las variables lingüísticas para la generación de reglas y las propias reglas; y el "motor de inferencia" qué cuenta con el mecanismo capaz de obtener la respuesta de salida del controlador en función a las reglas y a las entradas que se estén aplicando en un determinado momento.

Puesto que la mayoría de las aplicaciones cuenta con datos numéricos, debemos contar con un 'borrosificador' y un 'desborrosificador' para relacionar el mundo real con el sistema de control borroso. La borrosificación (*fuzzyfication*) consiste en asignar a cada entrada del universo de discurso, un término lingüístico representado por una función de pertenencia. El motor de inferencia cuenta con los siguientes pasos:

1. Calcular el grado de cumplimiento de cada antecedente en función de la entrada del sistema.
2. Composición de todos los antecedentes de cada regla.
3. Obtención del consecuente resultante de cada regla.
4. Combina los resultados de todas las reglas en un único conjunto borroso.
5. Se desborrosifica la salida en el caso de que sea necesario.

Existen múltiples formas de desborrosificar, pero las más comunes son:

- Método del centro de gravedad o del centroide.
- Método del centro de sumas.
- Método de la altura.
- Método de la máxima pertenencia.
- Método del centroide indexado.

En nuestro caso, elegiremos el método de centroide ya que genera variaciones suaves y continuas en los valores de salida. La salida cuantitativa se obtiene calculando el centro de gravedad del conjunto borroso de salida. Siendo '*D(y)*' el conjunto borroso de salida, su valor desborrosificado mediante este método se obtiene a partir de la ecuación:

$$y_o = \frac{\int y\mu_D(y)dy}{\int \mu_D(y)dy}$$

A continuación, en la Figura 2.9 podemos visualizar un ejemplo simple de cómo se genera un conjunto borroso de salida en un sistema que cuenta con dos reglas y un único antecedente:

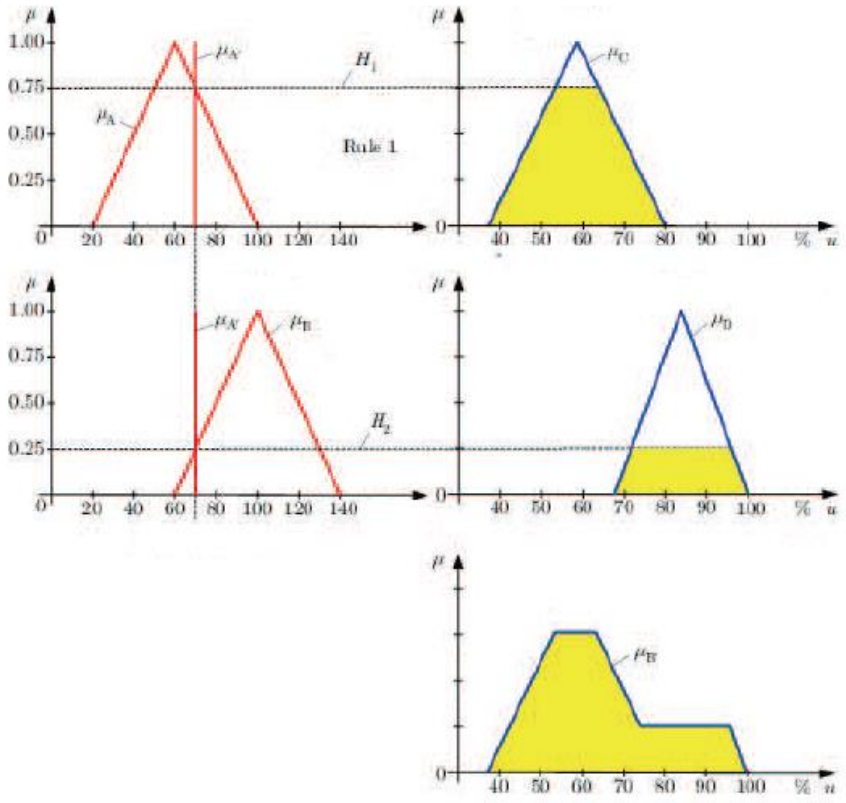


Figura 2.9: Desarrollo de una salida borrosa con dos reglas y un antecedente.

Las aplicaciones del control borroso abarcan campos industriales muy diversos:

- Intercambiadores de calor
- Hornos de cemento
- Secaderos de pulpa
- Grúas de contenedores
- Tratamientos de aguas
- Reactores nucleares
- Electrónica de consumo
- Robots móviles
- Operación de trenes
- Control de tráfico
- Control de vuelo

Respecto a la historia de la lógica difusa parece ser algo reciente, pero sus orígenes se remontan a los tiempos de Platón y Aristóteles, puesto que fueron los primeros en considerar que las cosas no tienen por qué ser siempre de un tipo o siempre dejar no serlo, sino que pueden existir escalas intermedias entre ambos extremos. Siglos después, David Hume e Immanuel Kant continuaron desarrollando esta idea. Ambos defendieron que el razonamiento se adquiere gracias a las vivencias a lo largo de la vida de las personas, por lo que el conocimiento podía variar y evolucionar con el transcurso del tiempo

Ya en el S.XX, Bertrand Russell estudió las contradicciones que presenta la lógica, debidas a la vaguedad con que se emplea el lenguaje. Por otro lado, Ludwing Wittgenstein desarrollo en un estudio las distintas acepciones que presentan las palabras, llegando a la conclusión de que una misma palabra expresa modos y maneras según sea empleada. En 1920 Jan Lukasiewicz desarrollo el primer prototipo de lógica borrosa en la que los conjuntos contaban con grados de pertenencia que oscilaban entre 0 y 1.

Aun así, Lofti Asier Zadeh es considerado el padre del término *borroso*, con su libro "Fuzzy Sets" ("Conjuntos borrosos") publicado en 1965. Su intención fue crear un formalismo para manejar eficientemente la imprecisión del razonamiento humano.

En 1974 Assilian y Mandani en Inglaterra, desarrollaron el primer controlador difuso diseñado para la máquina de vapor. La implantación real de un controlador borroso de este tipo se llevó a cabo por primera vez para una planta cementara en Dinamarca en 1980. A partir de ese momento, se empiezan a crear colaboraciones entre gobiernos, universidades e industrias. En 1987, la empresa Omron pone a la venta los primeros controladores difusos comerciales. Finalmente en el S.XXI, Takagi y Sugeno desarrollaron la primera aproximación para construir reglas *fuzzy* a partir de entrenamientos. Nuestro regulador *fuzzy* también contará con reglas para realizar el control de los sistemas. *(Véase referencias bibliográficas [7], [8], [9], [22] y [23]).*

2.2 Comparación Fuzzy-PID

Los controladores PID reúnen las ventajas de cada una de las tres acciones individuales de control (Proporcional-Integral-Derivativo). Los PID han demostrado ser robustos en muchísimas aplicaciones y por ello son los más utilizados en la industria. Su estructura es muy simple y ahí se encuentra su debilidad, dado que por ello queda limitado el rango de sistemas capaces de controlar satisfactoriamente. Estos sistemas deben de responder con salidas linealmente proporcionales.

Por otro lado, los controladores *fuzzy* no requieren conocimientos precisos del proceso, ni del modelo exacto para la realización del control. Es capaz de interpretar y responder ante expresiones y datos que no determinen con exactitud los valores de datos de entrada. La clave se encuentra en adaptar este tipo de lenguaje en cuantificadores para nuestras inferencias de entrada al controlador. Podríamos resumir que la utilización de la lógica difusa es aconsejable para procesos muy complejos (cuando se carece de un modelo matemático simple) o para procesos altamente no lineales, incluso cuando el ajuste de una variable produzca desajustes en otras.

Además, los '*fuzzy*' poseen un campo de aplicación cada vez más amplio, como comentaremos más adelante, por lo que continúa sometiéndose a numerosos estudios de desarrollo e investigación. Puesto que es conocido el

gran potencial que poseen los controladores *fuzzy*, se puede pronosticar impresionantes desarrollos para el futuro.

(Véase referencias bibliográficas [7], [8], [9] y [22]).

2.3 Differential Evolution: concepto e historia

Differential Evolution es un método de optimización multipunto y derivativo utilizado para resolver problemas de gran complejidad matemática. Esta disciplina tan poderosa pertenece a la categoría de algoritmos evolutivos. Se trata de un método basado en poblaciones, es decir, se utiliza una búsqueda en paralelo para reducir el tiempo de cálculo y evitar soluciones locales que no cumplan totalmente los requisitos.

Los algoritmos evolutivos son procedimientos de optimización y búsqueda de soluciones basados en axiomas de la evolución biológica. Este proceso parte de un conjunto de valores que actúa como posible solución del problema. Estos valores irán evolucionando, es decir, modificándose, en busca de una solución optimizada que resuelva el problema con la mayor satisfacción posible desde el punto de vista de la eficiencia y la calidad de los resultados. Esta técnica guarda un estrecho paralelismo con el "Principio de Selección Natural" de Charles Darwin (1809-1882) y el "Principio de Reproducción Genética" de Gregor Mendel (1822-1884). Según estos principios, sólo los individuos mejor adaptados, tienen mayor longevidad y por tanto mayor probabilidad de reproducción. Por otro lado, los individuos descendientes de estos primeros tendrán mayor posibilidad de transmitir sus códigos genéticos a las próximas generaciones. En resumen, los algoritmos evolutivos modelan el proceso de evolución como una sucesión de cambios habituales en el conjunto solución. El análogo biológico de este conjunto serían los cromosomas celulares. El espacio de soluciones posibles es explorado aplicando transformaciones a éstas soluciones candidatas, tal y como podemos observar en los propios organismos vivientes.

Las transformaciones cuentan con los siguientes pasos:

1. Inicialización: Se inaugura una población inicial para todas las variables que se desee optimizar. Únicamente debemos tener en cuenta los valores máximos y mínimos que debemos respetar atendiendo a la naturaleza de las variables. Una vez que se hayan elegido individuos aptos, deberemos alterarlos de forma aleatoria con la esperanza de mejorar las especificaciones para la siguiente generación. (Ver Figura 2.10)

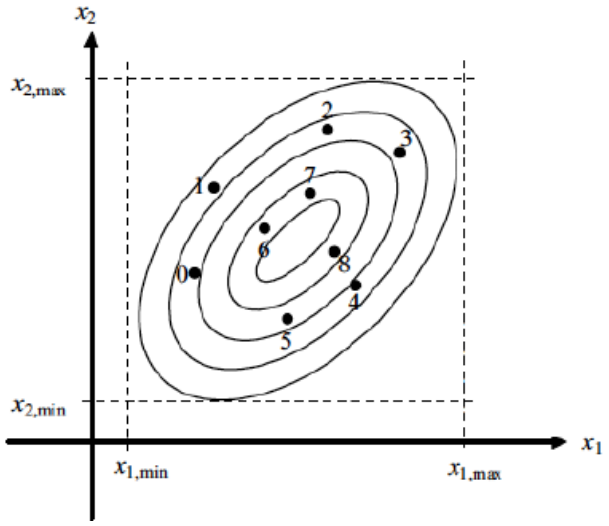


Figura 2.10: Inicialización de la población inicial en Differential Evolution.

2. Mutación: Este paso consiste en crear nuevos valores solución en forma de vector, recombinando de forma aleatoria los vectores originales o actuales que hubiese. Al igual que ocurre en los seres vivos al cambiar un gen por otro, la mutación de un algoritmo causa pequeñas variaciones en puntos concretos del circuito de control. Existen diferentes formas de mutar. Una forma habitual es crear una perturbación a partir de dos vectores de la población actual. Este paso cuenta con el factor de escala, 'F', que es un número real positivo que controla la tasa de población para crear vectores mutados o perturbados. (Ver Figura 2.11)

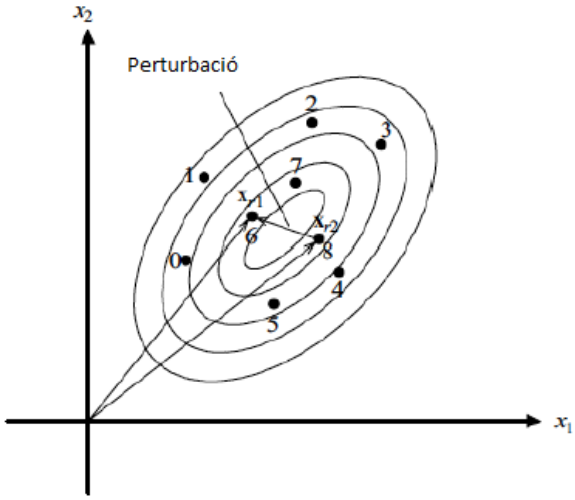


Figura 2.11: Mutación de la población en Differential Evolution.

3. Cruce: Una vez obtenidos los vectores perturbados, el cruce se efectúa de manera aleatoria utilizando los vectores originales y las perturbaciones, obteniendo los vectores de prueba. En este paso, encontramos la constante de cruce, 'CR'. Este valor es definido por el usuario en el rango [0, 1]. Después, se genera un número aleatorio para determinar qué individuo contribuye con un parámetro. Si el número aleatorio es menor o igual a CR, el parámetro a probar se hereda del vector mutante, de lo contrario se heredará el vector actual. Debido a esto, podemos definir CR como la probabilidad de que un parámetro de prueba sea el próximo heredero. (Ver Figura 2.12)

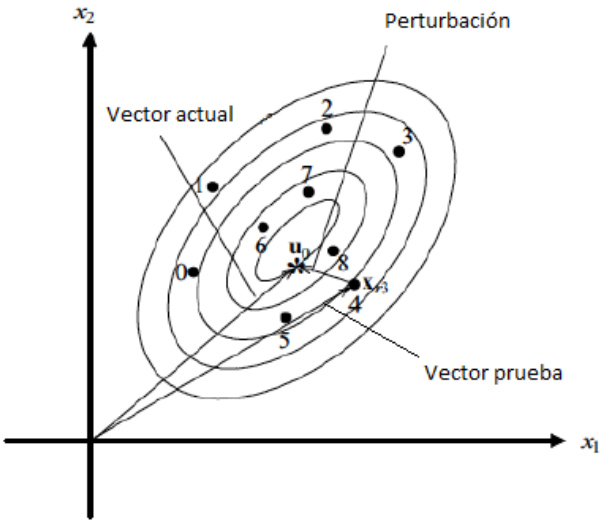


Figura 2.12: Cruce de la población en Differential Evolution.

4. Selección: Finalmente, se realiza la comparación entre los vectores de prueba mutados con los originales o actuales, de tal modo que el vector de la generación siguiente será aquel que consiga mejor valor en la función de coste. De esta manera, la población es capaz de evolucionar hacia una solución que desarrolle un resultado del problema lo más satisfactorio posible tras numerosas generaciones iteradas. Para nuestro caso particular, atenderemos la solución que desarrolle menores valores en el tiempo de estabilización (Ts), la sobreoscilación (Mp) y el tiempo de sobreoscialción (Tp) en nuestra función de costes. Existen diferentes técnicas de selección de los individuos mejor adaptados. Los más destacados son:

    a. Selección elitista: Se garantiza la elección de los individuos mejor adaptados. La mayoría de los algoritmos no utiliza un elitismo puro para evitar caer en optimizaciones locales en lugar de generales.
    b. Selección proporcional a la aptitud: Los individuos mejor adaptados tienen mayor probabilidad de ser seleccionados, pero no es seguro.
    c. Selección por rueda a rueda: La probabilidad que poseen los individuos para ser elegidos se encuentra de forma proporcional en la diferencia en las aptitudes que se consiguen en comparación con la que adquieren los individuos competidores.
    d. Selección por rango: Se le asigna a cada individuo un rango numérico basado en su aptitud. La selección se hace en base a este ranking. Este método puede evitar que los individuos bien adaptados del principio ganen dominancia y provoque falta de diversidad y dificulte encontrar una solución aceptable.
    e. Selección generacional: La descendencia de los individuos actuales toma el relevo. No se conservan nunca las generaciones pasadas.
    f. Selección por estado estacionario: Se conservan algunos individuos a lo largo de la optimización. En cada generación se permite reemplazar algunos de los individuos peor adaptados por los nuevos, si éstos fuesen más aptos.

Históricamente, Kenneth Price y Rainer Storn fueron los primeros en desarrollar la técnica de Evolución Diferencial. La primera publicación la realizaron en octubre de 1995 bajo el nombre original de "Differential Evolution -- a simple and efficient adaptive scheme for global optimization over continuos spaces" ("Evolución Diferencial – esquema simple y eficiente de optimización global en espacios continuos"). Este método estaba originalmente enfocado en resolver el problema de ajuste del polinomio de Tchebychev. En 1996, *Differential Evolution* fue presentado en un concurso que buscaba métodos de optimización de computación evolutiva, quedando como tercera propuesta más importante. Hoy día, continúa siendo una de las corrientes principales de la investigación en computación evolutiva. *(Véase referencias bibliográficas [10], [11], [12], [17], [18] y [19]).*

3 Diseño del controlador Fuzzy

En este apartado procederemos a explicar el diseño utilizado en el control de los sistemas. El regulador escogido es un controlador *Fuzzy*, también denominado de tipo borroso. La elección de este tipo de controlador se debe a su gran adaptabilidad al mundo real en el que vivimos, ya que es capaz de descifrar y responder ante expresiones que no determinen con exactitud los valores de datos de entrada. El ejemplo lingüístico de esta interpretación sería el uso de expresiones del tipo: *mucho, poco, mas, menos, muy, bastante, demasiado, tanto*… La clave se encuentra en adaptar este tipo de lenguaje en cuantificadores para nuestras inferencias de entrada al controlador.

Para llevar a cabo el control de los sistemas, montaremos un circuito de simulación utilizando la herramienta *'Simulik'* encontrada en el programa **MATLAB** versión R2013a. En el circuito acoplaremos una generación de señales tipo escalón (Step), un bloque representativo de la función de transferencia de los sistemas (Plant&Actuator), un bloque controlador *Fuzzy* (Fuzzy Logic Controller), 3 ganancias (Gain) destinadas a ser optimizadas por *Differential Evolution* (constante proporcional, constante derivativa y ganancia de salida del Fuzzy Logic Controller) y un osciloscopio (Scope) para visualizar la salida de los sistemas. Finalmente, nuestro circuito tendrá la forma representada en la Figura 3.1:

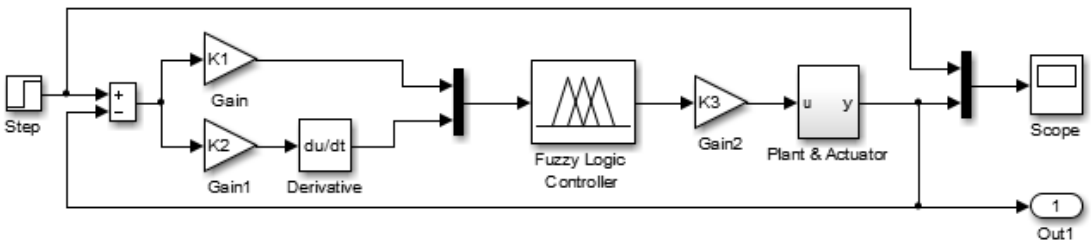


Figura 3.1: Circuito de simulación del control de sistemas.

En nuestro diseño de controlador borroso (*Fuzzy*), nos proponemos que se comporte de una manera equivalente al funcionamiento que tendría un Controlador Proporcional Derivativo (PD). La parte proporcional resulta ser el producto entre la señal de error y la constante proporcional para tratar que el error en estado estacionario se aproxime a cero. Sin embargo, existe un valor límite a partir del cual en algunos sistemas se consiguen valores superiores a los

deseados, fenómeno conocido como *sobreoscilación*. Por otro lado, la acción derivativa se produce por la variación en el tiempo del error entre el valor de consigna y la salida real del sistema. La función de la acción derivativa es mantener el error al mínimo valor posible gracias a que esta acción es capaz de adelantarse al comportamiento futuro del sistema. Si se produjese un error constante, y debido a esto el valor derivativo quedase anulado por completo, deberíamos recurrir a un controlador de tipo integral. Este último es capaz de disminuir o eliminar el error en estado estacionario provocado por la acción integral. Si tras realizar la optimización del control de los sistemas, obtuviéramos un error en estado estacionario superior al 5%, incorporaremos un controlador integral en paralelo al controlador *Fuzzy* para disminuir o eliminar este último error. El motivo por el cual el *Fuzzy* no incorpora la parte integral se debe a que el algoritmo de *Differential Evolution* que utilizaremos, el cual nos ha sido proporcionado por el tutor del proyecto Luis Garrido Bullón, sólo es capaz de optimizar 3 parámetros, los mismos que utiliza nuestro controlador *Fuzzy* equivalente al PD.

Para crear nuestro controlador *Fuzzy* teclearemos en Matlab el comando 'fuzzy' y de esta manera se nos abrirá una ventana estándar de FIS (Fuzzy Inference System) de tipo Mandani, como podemos observar en la Figura 3.2. En este tipo, los modelos tienen reglas donde los antecedentes y los consecuentes son proposiciones borrosas, con una entrada y una salida, pero nosotros deberemos ampliar el número de entradas a dos, que serán, por un lado el error absoluto (Error) cometido por el sistema en la salida respecto a al valor de consigna establecido, y por otro la derivada temporal de este error (Rate).

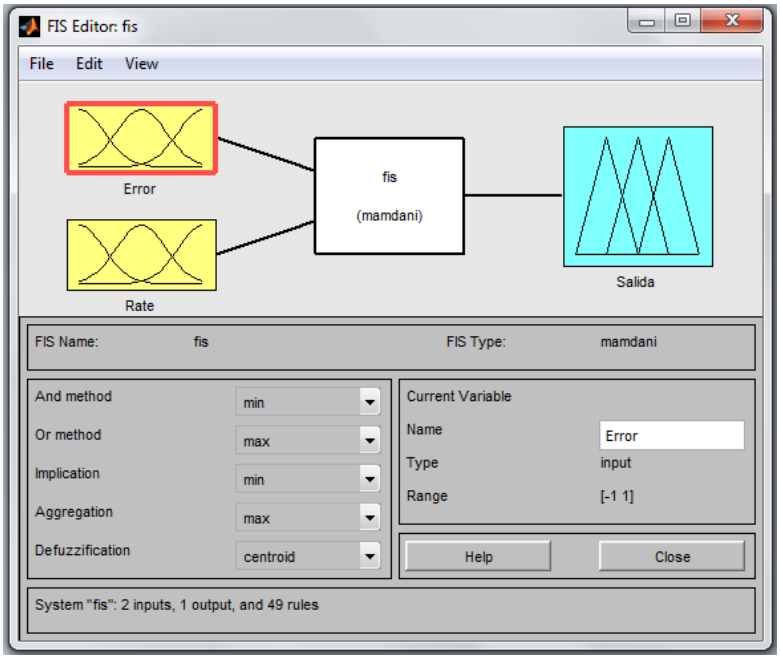


Figura 3.2: Ventana estándar Fuzzy Interface System.

El objetivo del diseño del controlador consiste en que sea capaz de crear una señal de salida desde el controlador que permita ajustar la salida del sistema de acuerdo a los requisitos que nos propusimos al inicio del proyecto. Como dijimos en la introducción, éstos son conseguir una sobreoscilaión máxima del 30% respecto del valor de consigna establecido en un tiempo máximo de 2.5 segundos y un tiempo máximo de estabilización de 5 segundos desde el arranque.

Para cumplir estos requisitos, diseñaremos una salida que dependa del error instantáneo que produzca el sistema y de la derivada temporal de este error. Para la clasificación tendremos que interesarnos por el valor absoluto de estas entradas y del signo de este valor. Por su parte, limitaremos tanto las entradas como la salida en un rango comprendido entre [-1, 1], por lo que la optimización de los parámetros deberá estar sujeta a esta condición. Para clasificar los márgenes utilizaremos 7 términos lingüísticos que serán: *negative big (bN),negative médium (mN), negative small (sN), zero (Z), positive small (sP), positive medim (mP) y positive big (bP*). Estos 7 términos en cada una de las 2 entradas proporcionan $7^2 = 49$ reglas lógicas que determinaran cuál de los 7 términos de la salida deberá actuar. En la Figura 3.3 podemos visualizar estas 49 reglas:

1. If (Error is Nb) and (Rate is Nb) then (Salida is Nb) (1)
2. If (Error is Nb) and (Rate is Nm) then (Salida is Nb) (1)
3. If (Error is Nb) and (Rate is Ns) then (Salida is Nb) (1)
4. If (Error is Nb) and (Rate is Z) then (Salida is Nb) (1)
5. If (Error is Nb) and (Rate is Ps) then (Salida is Nm) (1)
6. If (Error is Nb) and (Rate is Pm) then (Salida is Ns) (1)
7. If (Error is Nb) and (Rate is Pb) then (Salida is Z) (1)
8. If (Error is Nm) and (Rate is Nb) then (Salida is Nb) (1)
9. If (Error is Nm) and (Rate is Nm) then (Salida is Nb) (1)
10. If (Error is Nm) and (Rate is Ns) then (Salida is Nb) (1)
11. If (Error is Nm) and (Rate is Z) then (Salida is Nm) (1)
12. If (Error is Nm) and (Rate is Ps) then (Salida is Ns) (1)
13. If (Error is Nm) and (Rate is Pm) then (Salida is Z) (1)
14. If (Error is Nm) and (Rate is Pb) then (Salida is Ps) (1)
15. If (Error is Ns) and (Rate is Nb) then (Salida is Nb) (1)
16. If (Error is Ns) and (Rate is Nm) then (Salida is Nb) (1)
17. If (Error is Ns) and (Rate is Ns) then (Salida is Nm) (1)
18. If (Error is Ns) and (Rate is Z) then (Salida is Ns) (1)
19. If (Error is Ns) and (Rate is Ps) then (Salida is Z) (1)
20. If (Error is Ns) and (Rate is Pm) then (Salida is Ps) (1)
21. If (Error is Ns) and (Rate is Pb) then (Salida is Pm) (1)
22. If (Error is Z) and (Rate is Nb) then (Salida is Nb) (1)
23. If (Error is Z) and (Rate is Nm) then (Salida is Nm) (1)
24. If (Error is Z) and (Rate is Ns) then (Salida is Ns) (1)
25. If (Error is Z) and (Rate is Z) then (Salida is Z) (1)
26. If (Error is Z) and (Rate is Ps) then (Salida is Ps) (1)
27. If (Error is Z) and (Rate is Pm) then (Salida is Pm) (1)
28. If (Error is Z) and (Rate is Pb) then (Salida is Pb) (1)
29. If (Error is Ps) and (Rate is Nb) then (Salida is Nm) (1)
30. If (Error is Ps) and (Rate is Nm) then (Salida is Ns) (1)
31. If (Error is Ps) and (Rate is Ns) then (Salida is Z) (1)
32. If (Error is Ps) and (Rate is Z) then (Salida is Ps) (1)
33. If (Error is Ps) and (Rate is Ps) then (Salida is Pm) (1)
34. If (Error is Ps) and (Rate is Pm) then (Salida is Pb) (1)
35. If (Error is Ps) and (Rate is Pb) then (Salida is Pb) (1)
36. If (Error is Pm) and (Rate is Nb) then (Salida is Ns) (1)
37. If (Error is Pm) and (Rate is Nm) then (Salida is Z) (1)
38. If (Error is Pm) and (Rate is Ns) then (Salida is Ps) (1)
39. If (Error is Pm) and (Rate is Z) then (Salida is Pm) (1)
40. If (Error is Pm) and (Rate is Ps) then (Salida is Pb) (1)
41. If (Error is Pm) and (Rate is Pm) then (Salida is Pb) (1)
42. If (Error is Pm) and (Rate is Pb) then (Salida is Pb) (1)
43. If (Error is Pb) and (Rate is Nb) then (Salida is Z) (1)
44. If (Error is Pb) and (Rate is Nm) then (Salida is Ps) (1)
45. If (Error is Pb) and (Rate is Ns) then (Salida is Pm) (1)
46. If (Error is Pb) and (Rate is Z) then (Salida is Pb) (1)
47. If (Error is Pb) and (Rate is Ps) then (Salida is Pb) (1)
48. If (Error is Pb) and (Rate is Pm) then (Salida is Pb) (1)
49. If (Error is Pb) and (Rate is Pb) then (Salida is Pb) (1)

Figura 3.3: Reglas del controlador Fuzzy.

Dado que son muy latosas de leer, vamos a representarlas en la Tabla 3.1:

	Rate						
Error	nb	nm	ns	Z	ps	pm	pb
nb	nb	nb	nb	Nb	nm	ns	z
nm	nb	nb	nb	nm	ns	z	ps
ns	nb	nb	nm	ns	z	ps	pm
z	nb	nm	ns	z	ps	pm	pb
ps	nm	ns	z	ps	pm	pb	pb
pm	ns	z	ps	pm	pb	pb	pb
pb	z	ps	pm	pb	pb	pb	pb

Tabla 3.1: Reglas del controlador Fuzzy

El objetivo que nos planteamos es conseguir un controlador que sea capaz de responder ante una combinación de estas reglas a la entrada con la respuesta adecuada para un correcto control. Deberemos crear un área de equilibrio, cuando el error y la derivada del error consigan un valor aproximadamente nulo, en la que el controlador responda con una salida mínima para corregir levemente el desajuste que se esté cometiendo. Esta área estará en el centro de una superficie mayor y a medida que nos alejemos del centro, debiéndose a un aumento del error y/o de la derivada del error, se deberá responder con mayor énfasis en el sistema para corregir la discordancia. Para visualizar mejor este planteamiento, mostraremos el Surface tridimensional finalmente conseguido después de diseñar nuestro controlador *Fuzzy* en la Figura 3.4:

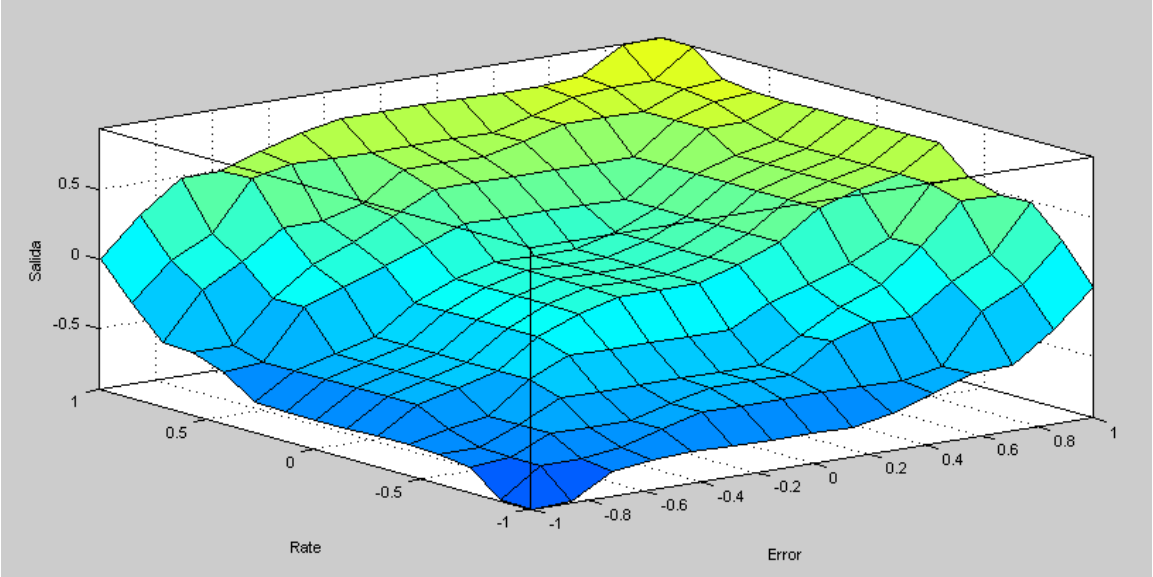


Figura 3.4: Surface del controlador Fuzzy utilizado.

Para la programación de los 49 términos lingüísticos utilizados tanto en las entradas (Error y Rate) y la salida, tuvimos que emplear la técnica de ensayo y error hasta conseguir que el Surface tuviese esta forma final. En las Figuras 3.5, 3.6 y 3.7 podremos observar el arquetipo de los términos lingüísticos. *(Véase referencias bibliográficas [3], [4], [7], [8] y [9]).*

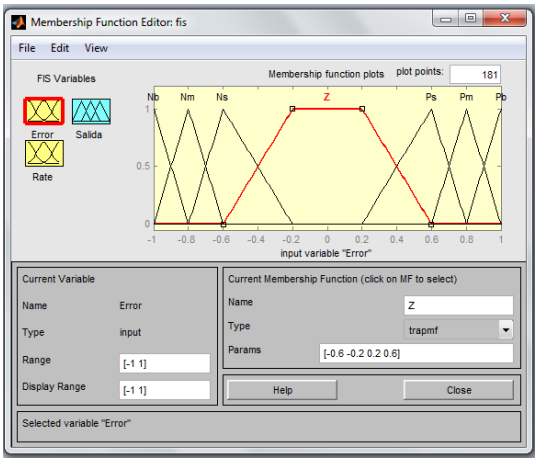


Figura 3.5: conjunto borroso de error.

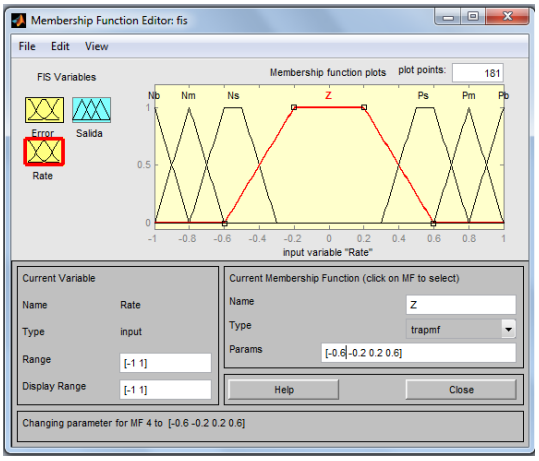


Figura 3.6: conjunto borroso de rate.

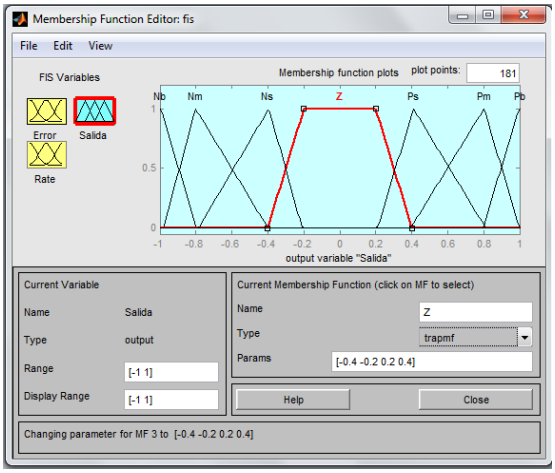


Figura 3.7: conjunto borroso de salida.

4 Implementación de Differential Evolution

En este trabajo trataremos de optimizar funciones artificiales, es decir, funciones sintéticas que no se corresponden con sistemas reales. Únicamente responden a modelos matemáticos aproximados, puesto que **nuestra finalidad es comprobar si *Differential Evolution* es capaz de optimizar los parámetros requeridos por un controlador borroso, a la entrada y la salida, en el control de sistemas cualesquiera, siempre que sean estables y de primer o segundo orden**. Este trabajo también podría haberse enfocado desde el punto de vista de que DE optimizase las reglas que utilice el Controlador *Fuzzy*, pero no es nuestro caso.

Debido al desconocimiento de los sistemas, deberemos emplear un método empírico de ensayo y error, para referenciar los distintos parámetros del algoritmo de optimización, si llegase a ser necesario, para lograr la mejor respuesta en la salida de los sistemas. Esto será necesario siempre y cuando tengamos dificultades para conseguir una salida que cumpla los requisitos especificados anteriormente.

Primeramente, elegiremos el método a seguir por el algoritmo evolutivo (strategy). Nuestra elección será *DE/rand/1/bin* ya que es la estrategia clásica y más usada en *Differential Evolution* por los expertos, por lo que en nuestro optimizador debemos otorgar 'strategy = 7'. El fundamento de esta estrategia es conservar el vector solución que mejor minimice la 'función objetivo' cuando la mutación de algoritmo se modifica con respecto a la información original. La función objetivo es conocida como 'función de coste', y en este trabajo fue desarrollada por el tutor de proyecto. El objetivo es disminuir al máximo posible el tiempo de estabilización de los sistemas (Ts) en el que el sistema alcance el 95% del valor de consigna y nunca salga de él; la sobreoscilación máxima (Mp); y el tiempo de sobreoscilación máxima (Tp). En la Figura 4.1 podemos observar donde encontrar estas especificaciones:

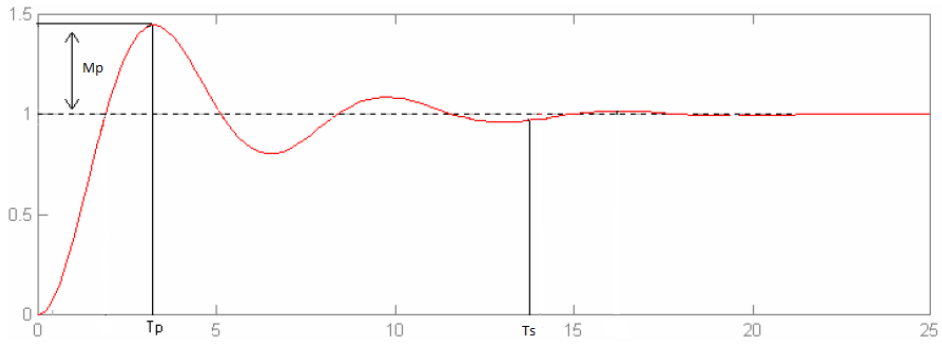


Figura 4.1: Salida estándar de un sistema subamortiguado.

El siguiente parámetro a definir será el número de variables a optimizar (D). Nuestro controlador borroso posee una constante proporcional (K1) y otra constante derivativa (K2) a la entrada del mismo, y por otro lado una ganancia (K3) a la salida, por lo que declaramos 'D = 3'. Los autores de manuales sobre *Differential Evolution* definen la población inicial (NP) como 10 veces mayor a los parámetros a optimizar, y recomiendan que la población no supere el valor 40, ya que empíricamente se ha demostrado que no mejora sustancialmente la convergencia independientemente del número de parámetros. Puesto que nosotros debemos optimizar 3 constantes, definiremos 'NP = 30' y seguiremos las recomendaciones de los autores.

Respecto al factor de amplificación diferencial para la mutación (F), aunque los autores recomiendan que su valor se encuentre en el intervalo [0, 2], se ha hallado de manera reciente que debe tener un valor comprendido entre [0.5, 1] para generar vectores solución que mejoren el comportamiento de convergencia significativamente, especialmente para sistemas ruidosos o posiblemente inestables. Por todo ello, definiremos 'F = 0.8' en nuestro optimizador. Por otro lado, la constante de cruce (CR) quedará definida en el intervalo [0, 1] según los manuales. Los autores recomiendan usar un valor alto para este parámetro, para así ayudar a mantener la diversidad de la población que va generando. Nuestra elección será declarar 'CR = 0.9'.

Dentro de los últimos parámetros a declarar en nuestro optimizador, cabe la pena destacar el error máximo (VTR) que el sistema debe cumplir una vez se ha logrado optimizar. Para protegernos de salidas insatisfactorias,

definimos este parámetro de orden muy pequeño. En nuestro caso, optamos por declarar 'VTR = 10^-6'. Además, debemos especificar el número de iteraciones máximas que nuestro optimizador deba generar, para acotar el tiempo que el ordenador deba emplear en calcular nuevos parámetros de control. De no ser así, el PC podría permanecer indefinidamente en proceso de optimización y jamás nos devolvería ningún resultado final. Para evitar esto, el código de programación se protege definiendo 200 iteraciones máximas en caso de no declarar otro valor inferior a éste. Experimentalmente, hemos comprobado que el algoritmo deja de disminuir la función de costes entre las 30 y las 40 iteraciones, por lo que definimos 'itermax = 40'.

Finalmente, debemos declarar un intervalo apropiado para la población inicial (XVmax y XVmin) antes de empezar a optimizarla. Debido al diseño del controlador borroso, las entradas a este deberán estar acotadas en el intervalo [0, 1] por lo que la constante proporcional deberá estar comprendida igualmente en este intervalo. Por otro lado, al tratarse de sistemas estables y poco ruidosos, sabemos que la constante derivativa deberá ser a priori de orden inferior a la proporcional. Por último, debido a que la salida del controlador borroso también se encuentra limitada por diseño en el intervalo [0, 1], la ganancia del controlador deberá ser de orden superior al resto de constantes para garantizar una señal suficientemente amplia en la entrada del sistema, y así lograr corregir el error que éste este cometiendo respecto de la referencia que estemos utilizando. Puesto que los intervalos máximo y mínimo se corresponden con tres parámetros constantes, XVmax y XVmin tienen forma de vector y quedan de esta manera: 'XVmin = [0, 0, 100]' y 'XVmax = [1, 0.1, 500]'.

Dentro de nuestro editor en dónde introduciremos todos estos parámetros, deberemos añadir la función 'gavec3', la cual es la responsable de minimizar la función de costes para la población determinada utilizando el algoritmo evolutivo en la correcta optimización de los sistemas. Es decir, hace evolucionar una población inicial estándar hacia los valores que consigan un coste menor total. El código de esta función podremos encontrarlo en: http://www.icsi.berkeley.edu/~storm/code.html. La función 'gavec3' devuelve siempre tres argumentos de salida: *bestmem* (vector de parámetros con la mejor solución), ***bestval*** **(mejor valor para la función objetivo o de coste)** y *nfeval* (número de evaluaciones de la función objetivo). Por otro lado, los argumentos de entrada a la función son los declarados anteriormente en este mismo

apartado entre otros. A parte de los ya mencionados, debemos añadir el nombre del archivo que contiene la función de costes a minimizar; un vector *'y'* de datos del problema, el cual deberá permanecer constante durante todo el proceso de minimización; y por último un indicador para marcar cada cuantas iteraciones se debe mostrar los resultados que se van logrando ('refresh = 10'). La Figura 4.2 muestra cómo quedaría finalmente nuestro editor de texto en Matlab para la optimización de los sistemas:

```
        VTR = 1.e-6;

        D = 3;

        XVmin = [0,0,100];
        XVmax = [1,0.1,500];

        y=[1 1 1];

        NP = 30;

        itermax = 40;

        F = 0.8;

        CR = 0.9;

        strategy = 7;

        refresh = 10;

optsim2;
pid0 = [1 1 1];

[x,f,nf] = gavec3('tracklsq',VTR,D,XVmin,XVmax,y,NP,itermax,F,CR,strategy,refresh);
K1 = x(1);
K3 = x(2);
K2 = x(3);
```

Figura 4.2: Editor de Matlab con los parámetros utilizados en la optimización.

El algoritmo evoluciona hacia una combinación de los tres parámetros (K1, K2 y K3) que minimiza la función de costes o función objetivo. Esta función viene definida en un archivo llamado 'tracklsq.m' y evalúa las diferencias entre la salida y la entrada, y la evolución hacia una mejor función de coste que mejore el control de los sistemas. Esta función también ha sido definida por el tutor de proyecto. La Figura 4.3 muestra cómo quedaría finalmente definida la función *tracklsq*:

```
function F = tracklsq(pid,y)

K1 = pid(1);
K3 = pid(2);
K2 = pid(3);


opt = simset('solver','ode8','SrcWorkspace','Current');
[tout,xout,yout] = sim('optsim2',[0 20],opt);


F=sum(abs(yout-1));
```

Figura 4.3: Función 'tracklsq'.

La función de costes mide el área comprendida entre la salida real del sistema y el valor de consigna que establezcamos. El algoritmo hará evolucionar la respuesta del sistema hacia una reducción del tiempo de estabilización (Ts), sobreoscialción máxima (Mp) y el tiempo en que se produzca la sobreoscilación máxima (Tp), y con ello a una minimización de esta área y del correspondiente valor ***bestval*** **(mejor valor para la función objetivo o de coste).** (Ver Figura 4.4)

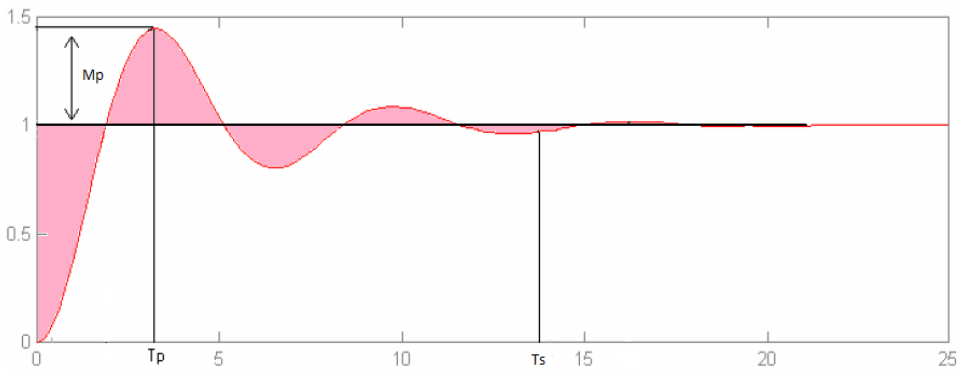


Figura 4.4: Área comprendida entre la salida real del sistema y el valor de consigna.

(Véase referencias bibliográficas [14], [16], y [21]).

5 Estudio de sistemas mediante función de transferencia

Para realizar el estudio sobre la respuesta temporal de los sistemas utilizaremos la herramienta "Simulink", provista en el programa Matlab, para reproducir el comportamiento con el que responda la salida de nuestro sistema, representado por una función de transferencia. El comando "sisotool" será el que utilizaremos para llevar a cabo este análisis, puesto que nos proporcionará tanto el lugar de las raíces como la representación gráfica de la salida del sistema. . *(Véase referencias bibliográficas [3], [4] y [6])*.

5.1 Sistema de primer orden

Comenzaremos la parte práctica del trabajo analizando y optimizando los sistemas más simples que podamos encontrar, es decir, los de primer orden. El ejemplo que usemos deberá cumplir que sea totalmente estable. Para asegurarnos este requerimiento, usaremos una función de transferencia que posea un único polo en el eje real en la parte negativa del lugar de las raíces. La función elegida es la siguiente:

$$G(s) = \frac{10}{s+4}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.1:

```
>> num=[10]

num =

    10

>> den=[1 4]

den =

     1     4

>> func_trans=tf(num,den)

func_trans =

   10
  -----
  s + 4

Continuous-time transfer function.

>>
```

Figura 5.1: Función de transferencia de sistema de primer orden.

Como ya hemos explicado, usaremos el comando "sisotool" para que Matlab nos devuelva tanto el lugar de las raíces como la salida de la función:

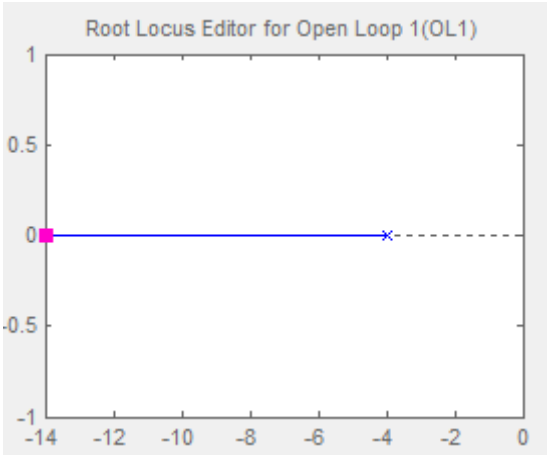


Figura 5.2: Lugar de las raíces de sistema de primer orden.

En la Figura 5.2 encontramos un único polo en $S = -4$, garantizando que la función es estable y de primer grado. A continuación, en la Figura 5.3 representamos la salida del ante una entrada escalón sin ningún tipo de control sobre el sistema, confirmando la estabilidad e éste:

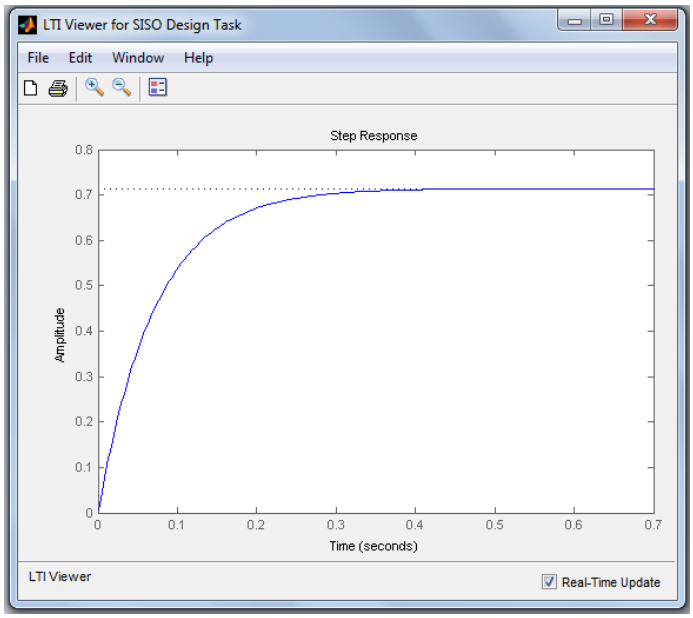


Figura 5.3: Salida del sistema de primer orden.

5.2. Sistemas de segundo orden

Continuando con la parte práctica del trabajo, nuestro siguiente ensayo constará de una función de segundo orden para aumentar la dificultar del proceso de optimización. Al igual que en el caso anterior, es totalmente imprescindible que las funciones que utilicemos sean estables para garantizar una respuesta coherente a nuestro objetivo.

Dentro de los tipos de funciones de segundo grado que puedan ser estables encontramos: sistemas subamortiguados, sistemas críticamente amortiguados y sistemas sobreamortiguados.

5.2.1. Sistema subamortiguado

Para asegurarnos que usamos un sistema estable de segundo orden subamortiguado, utilizaremos una función que posea un par de polos conjugados en la parte negativa del lugar de las raíces. La función elegida es la siguiente:

$$G(s) = \frac{10}{s^2 + s + 5}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.4:

```
>> num=[10]

num =

    10

>> den=[1 1 5]

den =

     1     1     5

>> func_trans=tf(num,den)

func_trans =

      10
  -----------
  s^2 + s + 5

Continuous-time transfer function.

>>
```

Figura 5.4: Función de transferencia de sistema de segundo orden subamortiguado.

Para asegurarnos que la función cuenta con un par de polos conjugados en la parte negativa del lugar de lar raíces, utilizaremos el comando "roots" de Matlab sobre el polinomio característico hallado en el denominador de la función como vemos en la Figura 5.5:

```
>> roots([1 1 5])

ans =

  -0.5000 + 2.1794i
  -0.5000 - 2.1794i

fx >>
```

Figura 5.5: Comando 'roots' de sistema de segundo orden subamortiguado.

Ahora usaremos el comando "sisotool", y Matlab nos devolverá el lugar de las raíces y la salida del sistema sin ningún tipo de control:

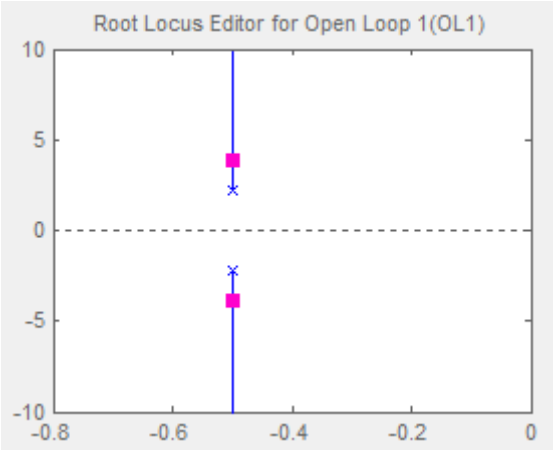


Figura 5.6: Lugar de las raíces de sistema de segundo orden subamortiguado.

En la Figura 5.6 encontramos un par de polos conjugados en $S_1 = -0.5 + 2.1794\mathbf{i}$ y $S_2 = -0.5 - 2.1794\mathbf{i}$, garantizando que la función es de segundo orden subamortiguada y estable. A continuación, en la Figura 5.7 representamos la salida de la función ante una entrada escalón sin ningún tipo de control sobre el sistema, confirmando la estabilidad de éste:

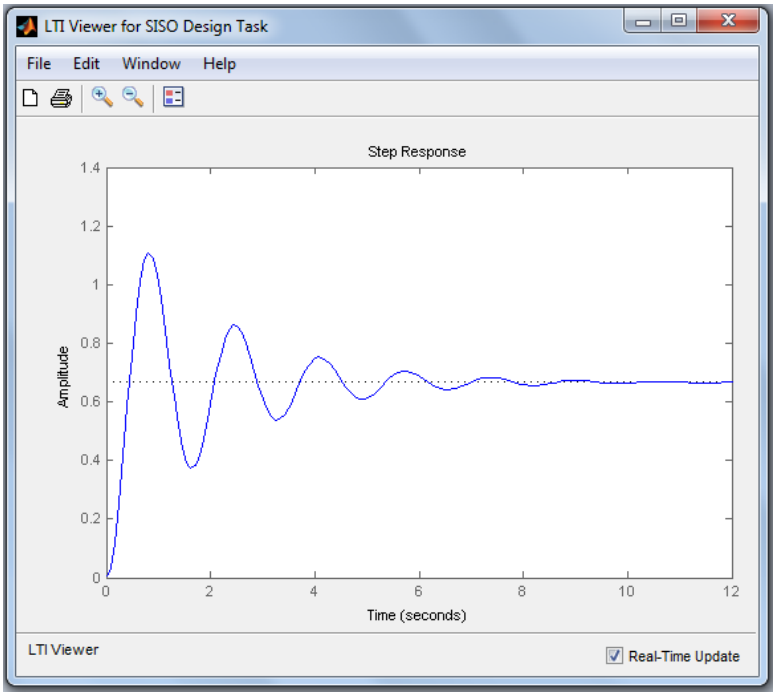


Figura 5.7: Salida del sistema de segundo orden subamortiguado

5.2.2. Sistema críticamente amortiguado

Seguidamente, analizaremos el siguiente tipo de sistema de segundo orden que pudiese ser estable. Se trata de un sistema críticamente amortiguado, el cual consta de un par de polos situados exactamente en la misma posición, o lo que es lo mismo un polo doble.

Para asegurar que se trate de un sistema críticamente amortiguado, este par de polos o polo doble debe situarse en el eje horizontal y al igual que siempre, deberá situarse en la parte negativa del lugar de las raíces para garantizar la estabilidad. La función elegida es la siguiente:

$$(s) = \frac{10}{(s+2)^2} = \frac{10}{s^2 + 4s + 4}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.8:

```
>> num=[10]

num =

    10

>> den=[1 4 4]

den =

     1     4     4

>> func_trans=tf(num,den)

func_trans =

        10
  -------------
  s^2 + 4 s + 4

Continuous-time transfer function.

>>
```

Figura 5.8: Función de transferencia de sistema de segundo orden críticamente amortiguado.

Ahora usaremos el comando "sisotool", y Matlab nos devolverá el lugar de las raíces y la salida del sistema sin ningún tipo de control:

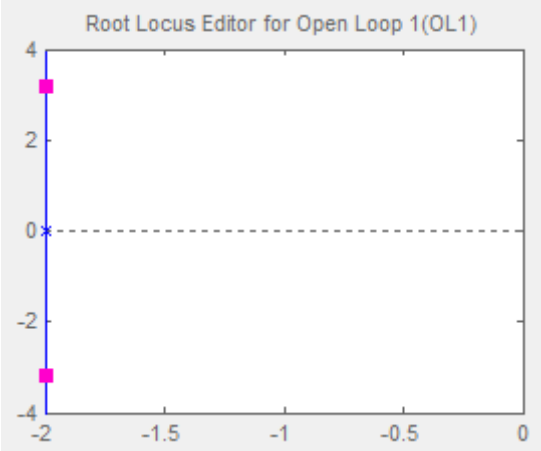


Figura 5.9: Lugar de las raíces de sistema de segundo orden críticamente amortiguado.

En la Figura 5.9 encontramos un par de polos o polo doble en $S_{12} = S_1 = S_2 = -2$ garantizando que la función es de segundo orden críticamente amortiguado y estable. A continuación, en la Figura 5.10 representamos la salida de la función ante una entrada escalón sin ningún tipo de control sobre el sistema, confirmando la estabilidad de éste:

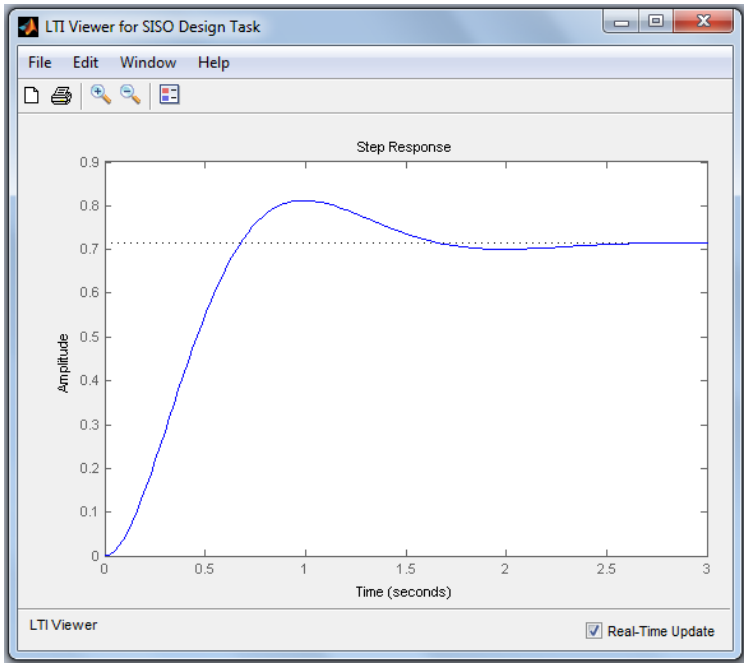


Figura 5.10: Salida del sistema de segundo orden críticamente amortiguado.

5.2.3. Sistema sobreamortiguado

Finalmente, analizaremos el último tipo de sistema de segundo orden que pudiese ser estable, es decir, el sistema sobreamortiguado, el cual consta de un par de polos situados en el eje horizontal del lugar de las raíces pero ocupando sitios diferentes.

Para asegurar que se trate de un sistema estable, este par de polos debe situarse al igual que siempre en la parte negativa del lugar de las raíces para garantizar la estabilidad. La función escogida es la siguiente:

$$G(s) = \frac{10}{(s+2)*(s+3)} = \frac{10}{s^2+5s+6}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.11:

```
>> num=[10]

num =

    10

>> den=[1 5 6]

den =

     1     5     6

>> func_trans=tf(num,den)

func_trans =

        10
  -------------
  s^2 + 5 s + 6

Continuous-time transfer function.

fx >>
```

Figura 5.11: Función de transferencia de sistema de segundo orden sobreamortiguado.

Utilizando el comando "sisotool", Matlab devolverá el lugar de las raíces y la salida del sistema sin ningún tipo de control:

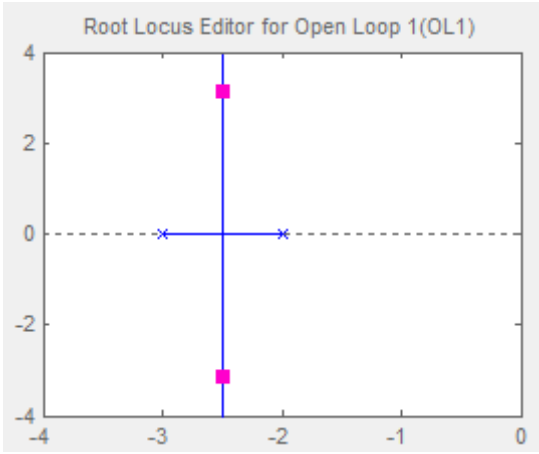


Figura 5.12: Lugar de las raíces de sistema de segundo orden sobreamortiguado.

En la Figura 5.12 encontramos un par de polos en $S_1 = -3$ y $S_2 = -2$ garantizando que la función es de segundo orden sobreamortiguada y estable. A continuación, en la Figura 5.13 representamos la salida de la función ante una entrada escalón sin ningún tipo de control sobre el sistema, confirmando la estabilidad de éste:

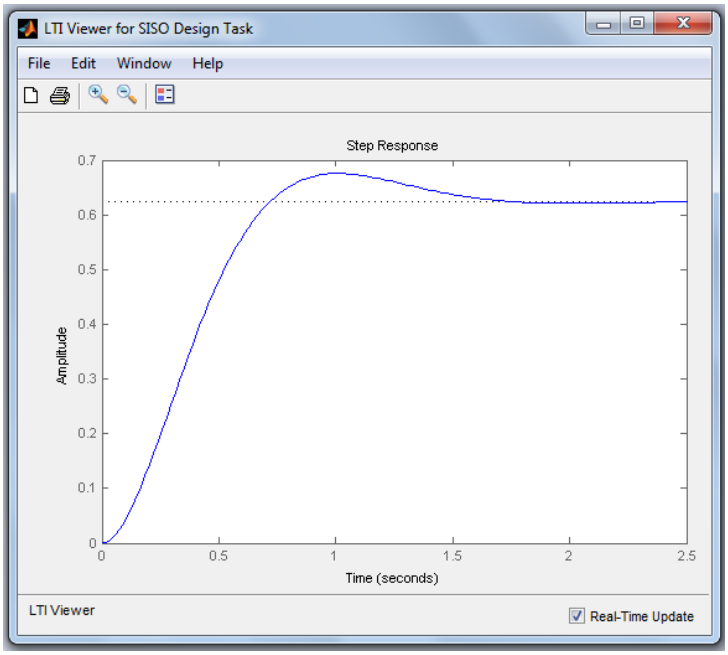


Figura 5.13: Salida del sistema de segundo orden sobreamortiguado.

5.2.4. Sistema con polo en el origen del lugar de las raíces

A continuación en este apartado de trabajo, procederemos a estudiar sistemas de segundo orden que contengan polos localizados en posiciones críticas en el lugar de las raíces. Pondremos a prueba el límite de control que sea capaz de desarrollar nuestro sistema de control. En especial, nos centraremos en estudiar los resultados de funciones que contengan un polo en el origen y un polo inestable.

La función elegida que cuente con un polo en el origen es la siguiente:

$$G(s) = \frac{10}{s * (s + 0.1)} = \frac{10}{s^2 + 0.1s}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.14:

```
>> num=[10]

num =

    10

>> den=[1 0.1 0]

den =

    1.0000    0.1000         0

>> func_trans=tf(num,den)

func_trans =

       10
  -----------
  s^2 + 0.1 s

Continuous-time transfer function.

>>
```

Figura 5.14: Función de transferencia de sistema de segundo orden con polo en el origen.

Para asegurarnos que la función cuenta con un polo en el origen, utilizaremos el comando "roots" de Matlab sobre el polinomio característico hallado en el denominador de la función como vemos en la Figura 5.15:

```
>> roots([1 0.1 0])

ans =

         0
   -0.1000

fx >>
```

Figura 5.15: Comando 'roots' de sistema de segundo orden con polo en el origen.

Ahora usaremos el comando "sisotool", y Matlab nos devolverá el lugar de las raíces y la salida del sistema sin ningún tipo de control:

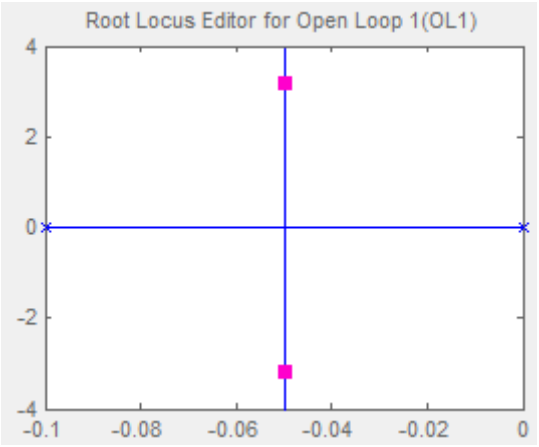


Figura 5.16 Lugar de las raíces de sistema de segundo orden con polo en el origen.

En la Figura 5.16 encontramos un de polo en el origen, $S_1 = 0$ y otro en $S_2 = -0.1$. A continuación, en la Figura 5.17 representamos la salida de la función ante una entrada escalón sin ningún tipo de control sobre el sistema:

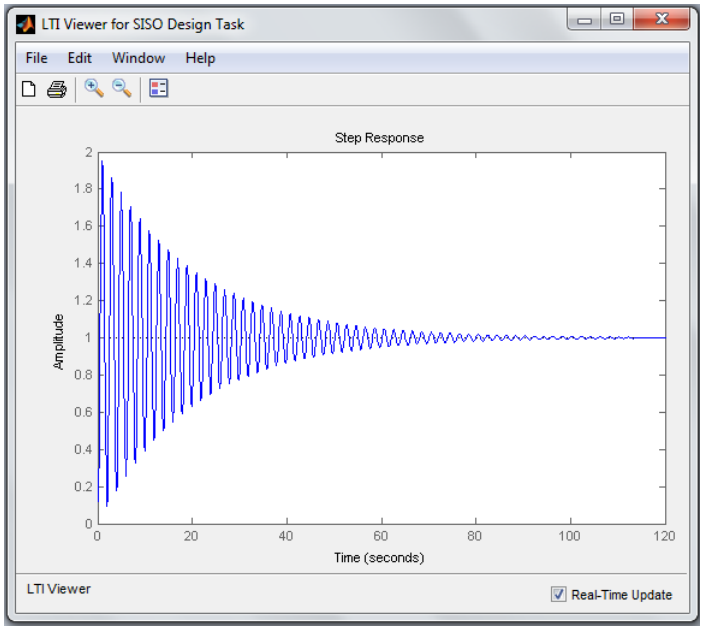


Figura 5.17: Salida del sistema de segundo orden con polo en el origen.

Podemos observar una gran inestabilidad en la puesta en marcha del sistema al mismo tiempo que logra una progresiva estabilidad. Pasado el minuto y medio, consigue un equilibrio libre de perturbaciones.

5.2.5. Sistema con polo inestable

Para lograr una función inestable, debemos situar uno o más polos situados en el semiplano positivos del lugar de las raíces. La función elegida es la siguiente:

$$G(s) = \frac{10}{(s+2) * (s-2)} = \frac{10}{s^2 - 4}$$

La cual declararemos en Matlab, tal como apreciamos en la Figura 5.18:

```
>> num=[10]

num =

    10

>> den=[1 0 -4]

den =

     1     0    -4

>> func_trans=tf(num,den)

func_trans =

    10
  -------
  s^2 - 4

Continuous-time transfer function.

>>
```

Figura 5.18: Función de transferencia de sistema de segundo orden con polo inestable.

Para asegurarnos que la función cuenta con un polo inestable, utilizaremos el comando "roots" de Matlab sobre el polinomio característico hallado en el denominador de la función como vemos en la Figura 5.19:

```
>> roots([1 0 -4])

ans =

    2.0000
   -2.0000

>>
```

Figura 5.19: Comando 'roots' de sistema de segundo orden con polo inestable.

Ahora usaremos el comando "sisotool", y Matlab nos devolverá el lugar de las raíces y la salida del sistema sin ningún tipo de control:

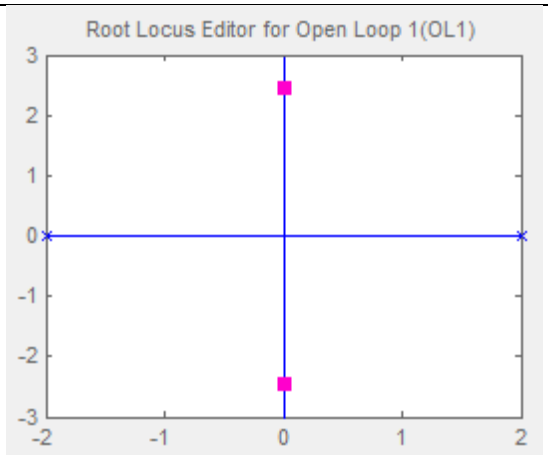


Figura 5.20 Lugar de las raíces de sistema de segundo orden con polo inestable.

En la Figura 5.20 encontramos un polo inestable en $S_1 = 2$ y otro polo estable en $S_2 = -2$. A continuación, en la Figura 5.21 representamos la salida de la función ante una entrada escalón sin ningún tipo de control sobre el sistema:

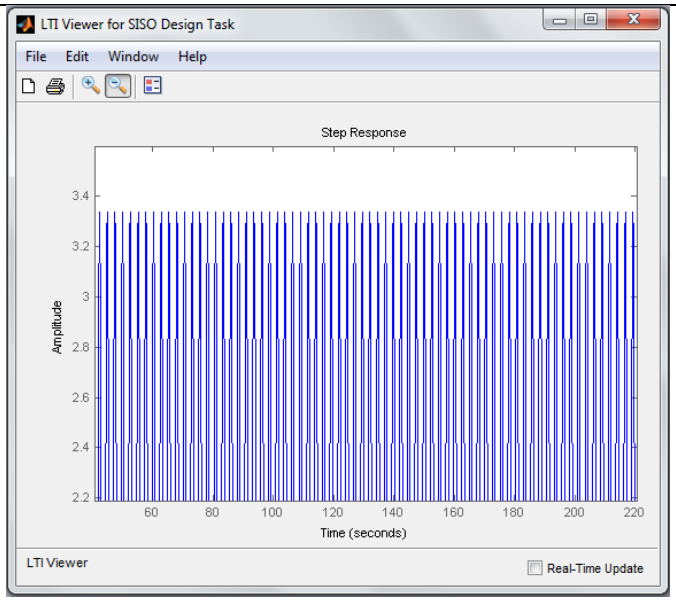


Figura 5.21: Salida del sistema de segundo orden con polo inestable.

Podemos observar una gran inestabilidad en la totalidad de la respuesta del sistema, como consecuencia al desequilibrio originado por el polo inestable.

6 Análisis de los resultados experimentales

6.1 Sistema de primer orden

Tras la optimización, mediante *Differential Evolution*, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de primer orden, Matlab nos devuelve las estimaciones de la Tabla 6.1:

Constante proporcional (K1)	0.1777
Constante derivativa (K2)	0.0479
Constante de ganancia (K3)	135.4496

Tabla 6.1: Constantes optimizadas del sistema de primer orden.

Atendiendo a estos valores, podemos observar como el valor de la constante proporcional es de orden inferior a la unidad. Esto es debido a que en el diseño del controlador borroso especificamos que entradas debían estar comprendidas entre -1 y +1, por lo que K1 deberá reducir el valor del error cometido por el sistema de manera proporcional hasta alcanzar este requisito.

Por otro lado, el control derivativo es imprescindible ya que existe un retardo temporal entre la salida y su repercusión final en el sistema a controlar. Cuando la acción derivativa es elevada se debe a que existe inestabilidad en el proceso. Experimentalmente hemos obtenido una estimación de K2 de orden menor a K1, y en consecuencia deducimos que la salida es capaz de oscilar entorno al punto de referencia en un corto espacio de tiempo.

Por último, debemos comentar que el valor tan elevado que obtenemos de la ganancia, en comparación con las anteriores constantes, también se debe al diseño del controlador borroso, ya que éste tiene una salida limitada entre -1 y +1, por lo que deberemos aumentar proporcionalmente dicha salida antes de que sea entregada en el sistema a controlar.

El resultado visual de nuestro experimento queda ilustrado en la Figura 6.1:

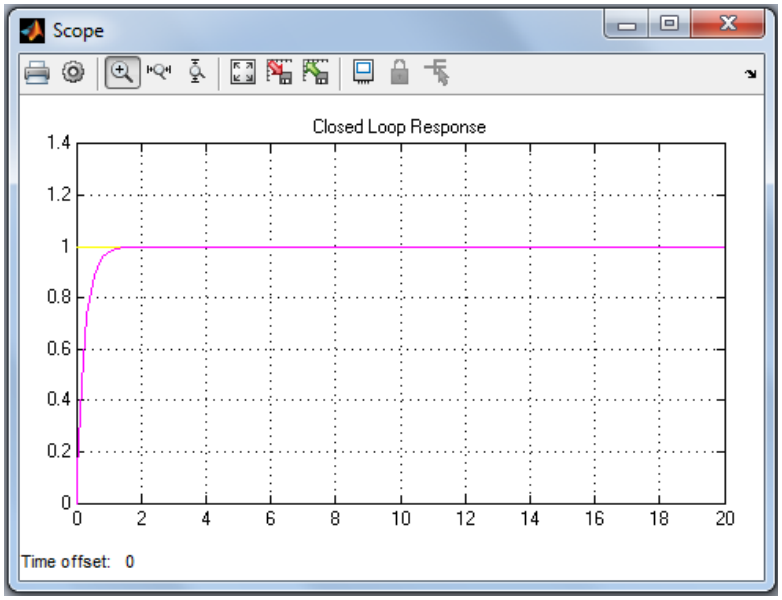


Figura 6.1 Salida optimizada del sistema de primer orden.

Para analizar mejor los resultados, ampliaremos la zona límite entre la respuesta transitoria y la zona de régimen permanente: (Ver Figura 6.2)

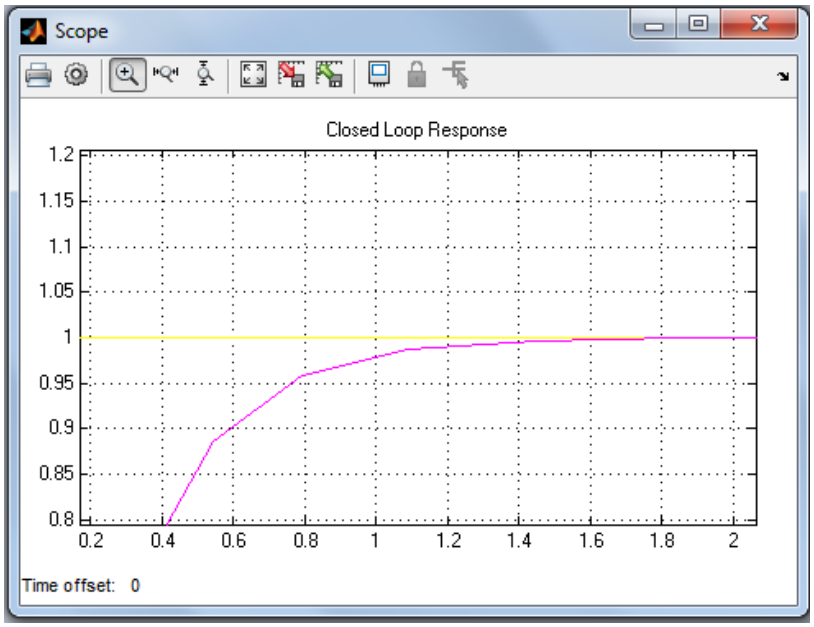


Figura 6.2 Salida optimizada ampliada del sistema de primer orden.

Ahora sí podemos observar en profundidad los resultados experimentales (Ver Tabla 6.2) para comprobar si el sistema controlado cumple las especificaciones exigidas al principio del trabajo.

PARÁMETROS	SISTEMA CONTROLADO
Tiempo de establecimiento, Ts	0.75 segundos
Tiempo de sobreoscilaión, Tp	-
Sobreoscilación, Mp (%)	0%

Tabla 6.2: Solución del sistema de primer orden.

Atendiendo a estos resultados, comprobamos que el tiempo de estabilización es muy inferior a los 5 segundos que nos propusimos como máximo, y por otro lado, encontramos una total ausencia de sobreoscilación.

En función a esta comparativa, concluimos que el sistema de control se ha optimizado de manera satisfactoria ante los requisitos impuestos.

6.2 Sistemas de segundo orden

6.2.1 Sistema subamortiguado

Aprovecharemos que la mayoría de los sistemas en ingeniería son subamortiguado para mostrar la evolución que desarrolla *Differential Evolution* en la optimización de los parámetros a lo largo de las 40 iteraciones que implementamos.

A las 10 iteraciones el sistema desarrolla la salida visualizada en la Figura 6.3:

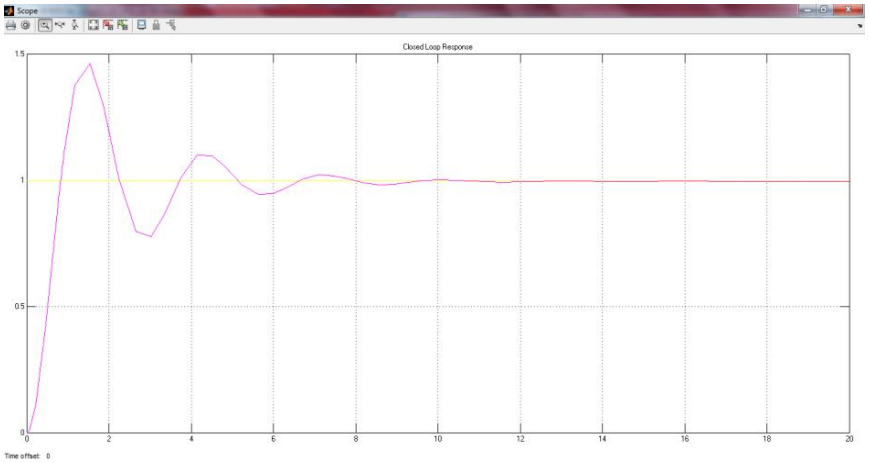


Figura 6.3: Salida del sistema subamortiguado a las 10 iteraciones.

Observamos una sobreoscilación máxima del 46% a los 1,55 segundos y un tiempo de estabilización de 6 segundos. La función de costes devuelve un valor final, *bestval*, de 1,904166.

20 iteraciones después la sobreoscilación máxima desciende al 18% a los 1,8 segundos y un tiempo de estabilización de 4,7 segundos. La función de costes también logra descender a 1,776131. La salida la mostramos en la Figura 6.4:

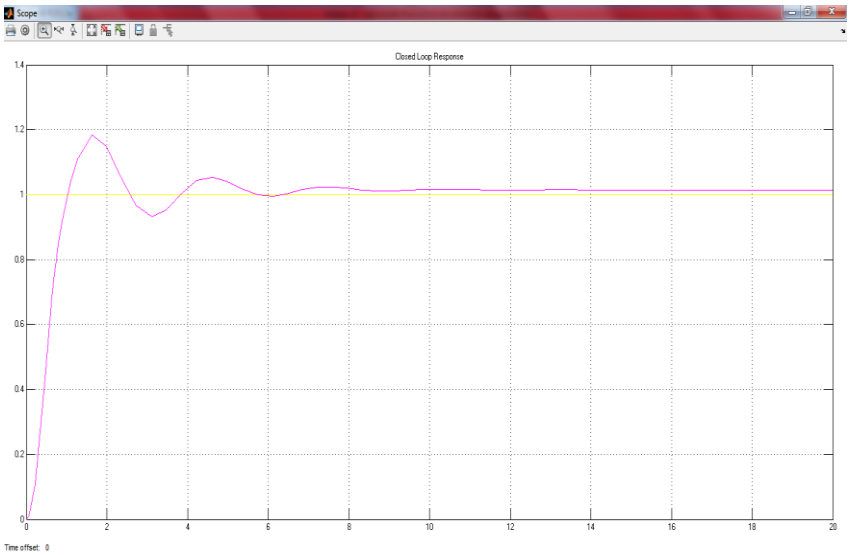


Figura 6.4: Salida del sistema subamortiguado a las 20 iteraciones

Seguidamente, 30 iteraciones después la sobreoscilación máxima empeora hasta el 50% a los 1,6 segundos. El tiempo de estabilización también empeora, ya que aumenta a los 5,8 segundos. Aun así, podemos asegurar que estos datos en conjunto son mejorados respecto a los anteriores, ya que el parámetro *bestval* disminuye a 1,300222. La salida la mostramos en la Figura 6.5:

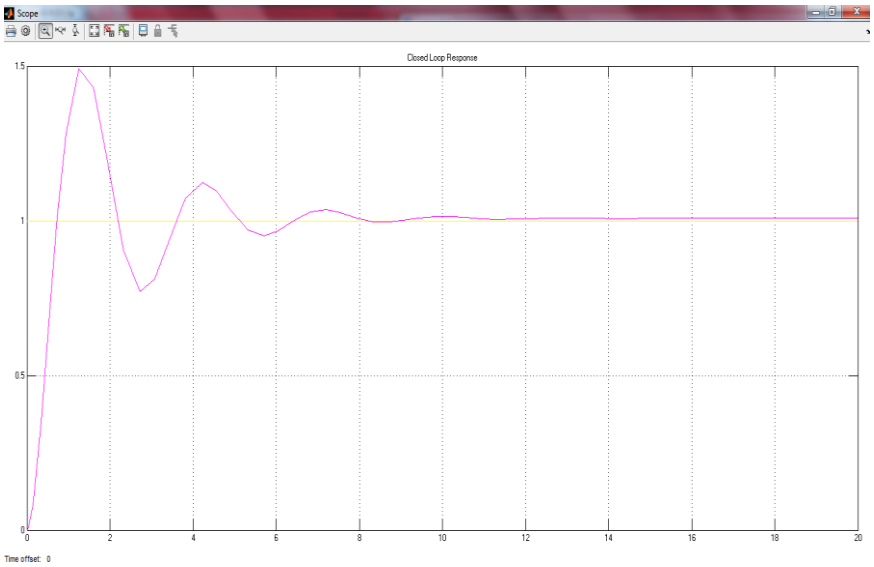


Figura 6.5: Salida del sistema subamortiguado a las 30 iteraciones

Tras la optimización final a las 40 iteraciones, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de segundo orden subamortiguado, Matlab nos devuelve las estimaciones de la Tabla 6.3:

Constante proporcional (K1)	0.1420
Constante derivativa (K2)	0.2304
Constante de ganancia (K3)	169.3873

Tabla 6.3: Constantes optimizadas del sistema de segundo orden subamortiguado.

En esta ocasión, observamos que la constante proporcional continua siendo de orden inferior a la unidad debido al diseño del controlador borroso que se explicó en el apartado anterior.

Por otro lado, esta vez el control derivativo resulta mayor que el proporcional. Esto se debe a que este sistema posee mayor inestabilidad y con ello mayor tiempo de respuesta transitoria que deberá ser reducida mediante el control borroso.

Por último, el valor de la ganancia continua siendo elevado debido también al diseño del controlador, como ya se explicó igualmente en el apartado anterior.

El resultado visual de nuestro experimento queda ilustrado a en la Figura 6.6:

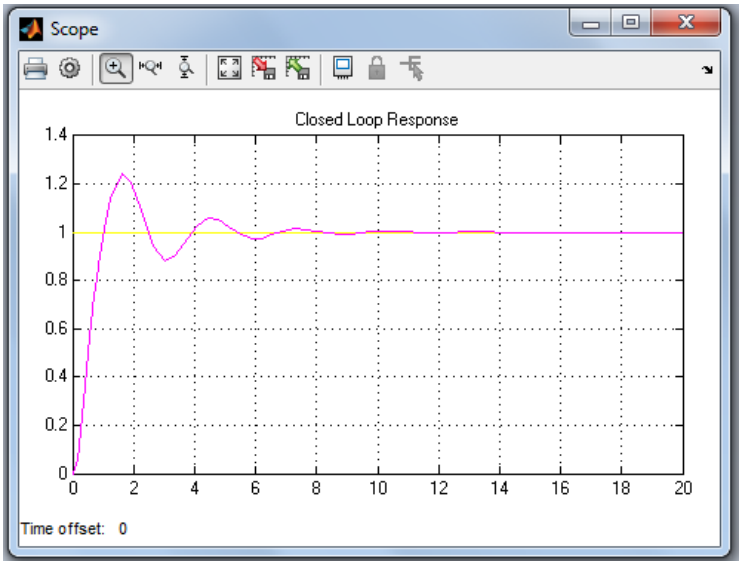


Figura 6.6: Salida optimizada del sistema de segundo orden subamortiguado.

Para analizar mejor los resultados, ampliaremos la zona límite entre la respuesta transitoria y la zona de régimen permanente en la Figura 6.7:

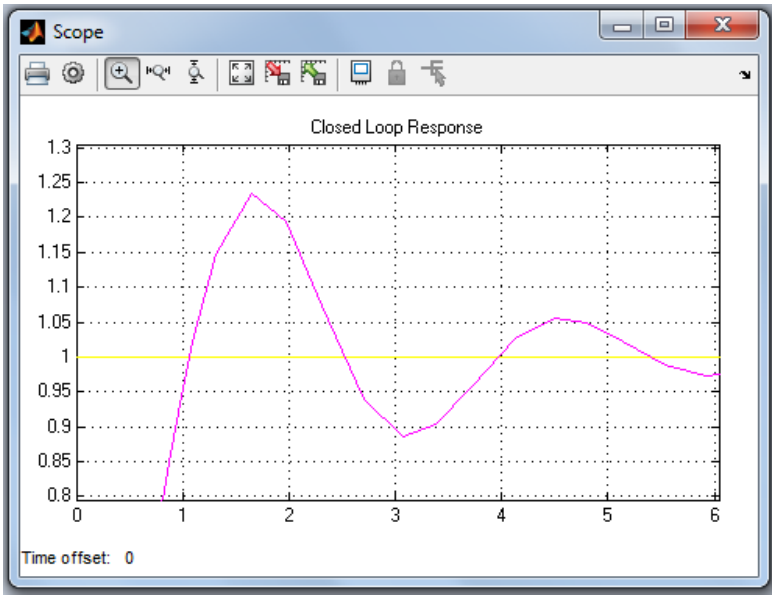


Figura 6.7: Salida optimizada ampliada del sistema de segundo orden subamortiguado

Ahora podemos observar en profundidad los resultados experimentales (Ver Tabla 6.4) para comprobar si el sistema controlado cumple las especificaciones exigidas al principio del trabajo.

PARÁMETROS	SISTEMA CONTROLADO
Tiempo de establecimiento, Ts	4.5 segundos
Tiempo de sobreoscilaión, Tp	1.6 segundos
Sobreoscilaión, Mp (%)	24%

Tabla 6.4: Solución del sistema de segundo orden subamortiguado.

Atendiendo a estos resultados, comprobamos que el tiempo de estabilización es levemente inferior a los 5 segundos que nos propusimos en las especificaciones, pero igualmente menor, por lo que consideramos que esta respuesta es apropiada. Por otro lado, podemos observar como la sobreoscilación máxima se da a los 1.6 segundos del arranque y alcanza un valor del 24% respecto a la referencia. Puesto que al comienzo impusimos que la sobreoscilación no debería pasar del 30% del valor final en un tiempo máximo de 2.5 segundos, valoramos que estas especificaciones resultan ser suficientemente convenientes. La función de costes devuelve finalmente el valor 1,242697182.

En función a esta comparativa, concluimos que el sistema de control se ha optimizado de manera satisfactoria ante los requisitos impuestos.

Para analizar en profundidad el desarrollo logrado por *Differential Evolution*, mostraremos cómo han evolucionado los parámetros 'Ts', 'Mp' y 'Tp' a lo largo de las 40 iteraciones en las Figuras 6.8, 6.9 y 6.10.

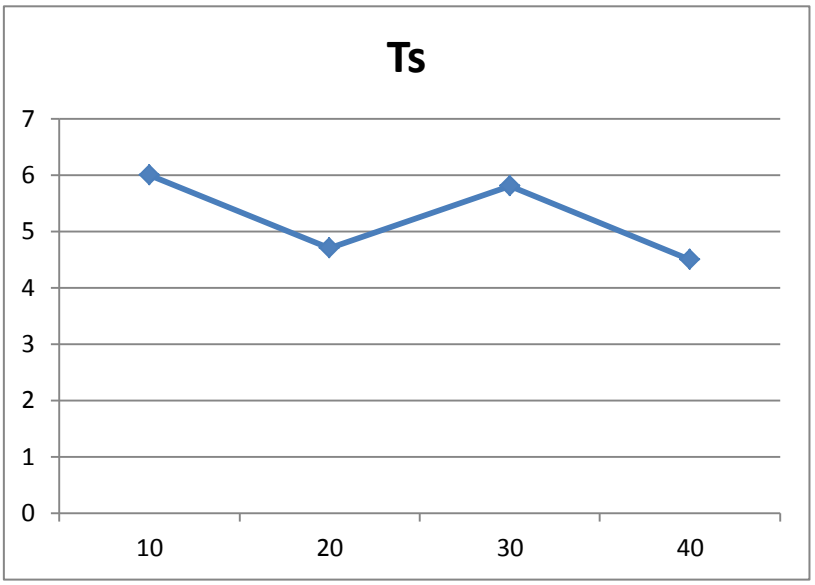


Figura 6.8: Evolución del tiempo de estabilización en sistema subamortiguado.

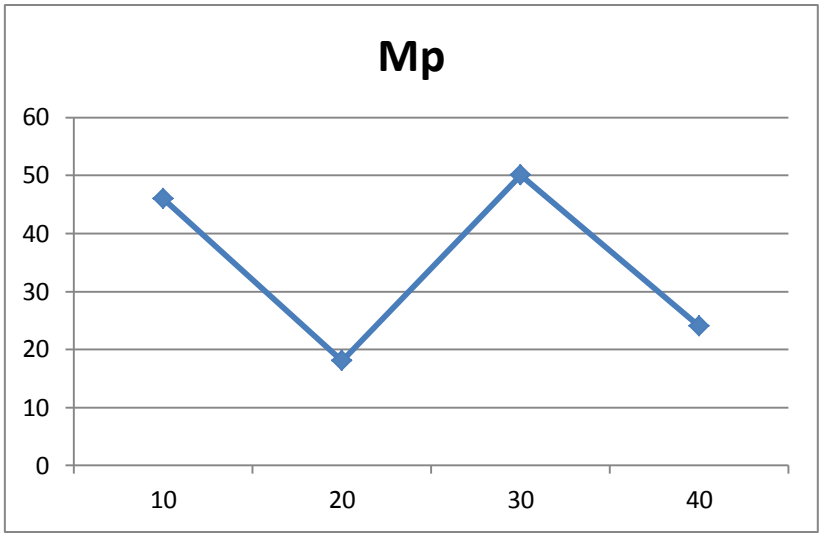


Figura 6.9: Evolución de la sobreoscilación máxima en sistema subamortiguado.

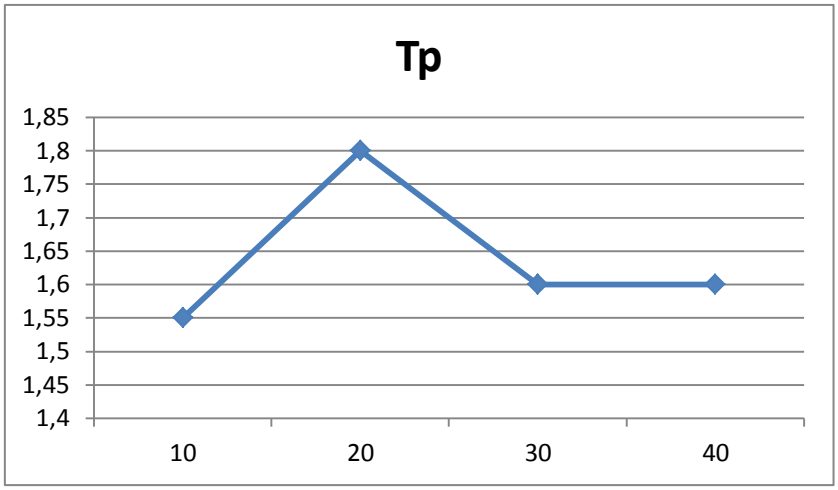


Figura 6.10: Evolución del tiempo de la sobreoscilación máxima en sistema subamortiguado.

Observando estas gráficas, vemos como la sobreoscilación máxima y el tiempo en que se produce están relacionados inversamente. Esto es, cuando logramos disminuir la sobreoscilación máxima se debe a que se produce de forma más tardía en el tiempo, e inversamente, cuando disminuimos este tiempo de sobreoscilación máxima es debido a que la sobreoscilación aumenta. A pesar de ello, podemos a asegurar que en conjunto estos parámetros evolucionan favorablemente en cada iteración, ya que el valor devuelvo por la función de costes guarda una tendencia descendente como vemos en la Figura 6.11:

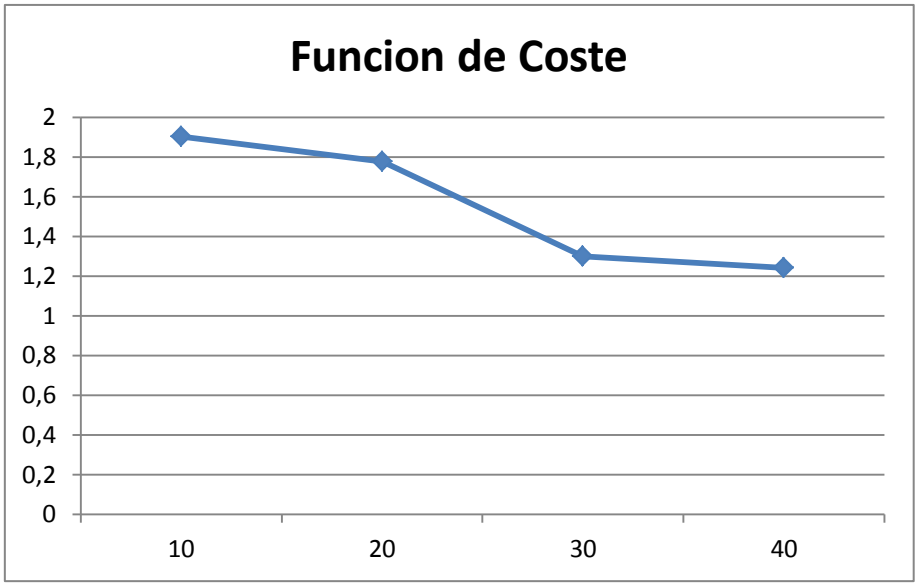


Figura 6.11: Evolución de la función de costes en sistema subamortiguado.

6.2.2 Sistema críticamente amortiguado

Tras la optimización, mediante *Differential Evolution*, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de segundo orden críticamente amortiguado, Matlab nos devuelve las estimaciones en la Tabla 6.5:

Constante proporcional (K1)	0.2110
Constante derivativa (K2)	0.1597
Constante de ganancia (K3)	135.1479

Tabla 6.5: Constantes parametrizadas del sistema de segundo orden críticamente amortiguado

Para este sistema, la constante proporcional continua siendo de orden inferior a la unidad debido al diseño. Esta vez, el control derivativo resulta menor que el proporcional, lo que nos hace pensar en una leve inestabilidad en el sistema y con ello mayor tiempo de respuesta transitoria acotado.

Por último, el valor de la ganancia continua siendo elevado para lograr mandar una señal suficientemente amplia al sistema para corregir el error que éste esté cometiendo.

El resultado visual de nuestro experimento queda ilustrado en l Figura 6.12:

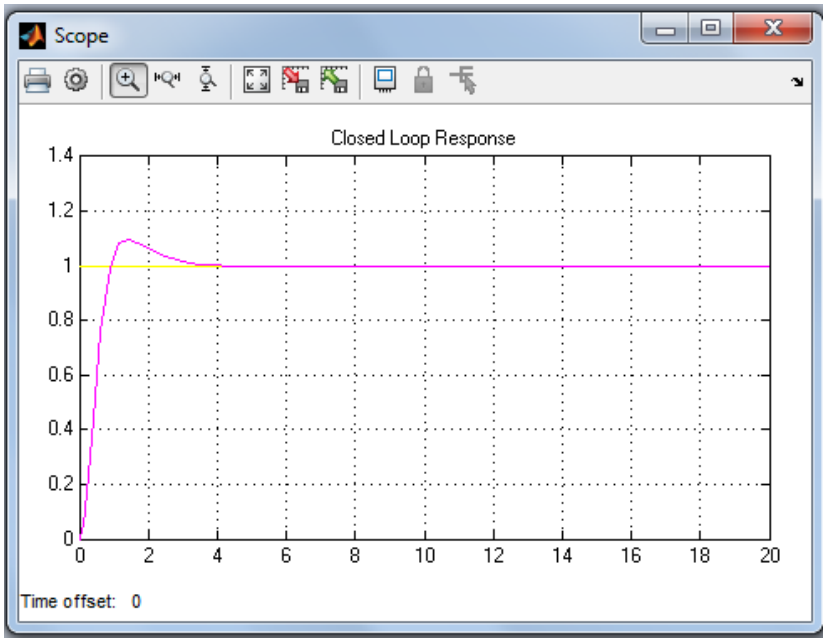


Figura 6.12: Salida optimizada del sistema de segundo orden críticamente amortiguado.

Para analizar mejor los resultados, ampliaremos la zona límite entre la respuesta transitoria y la zona de régimen permanente en la Figura 6.13:

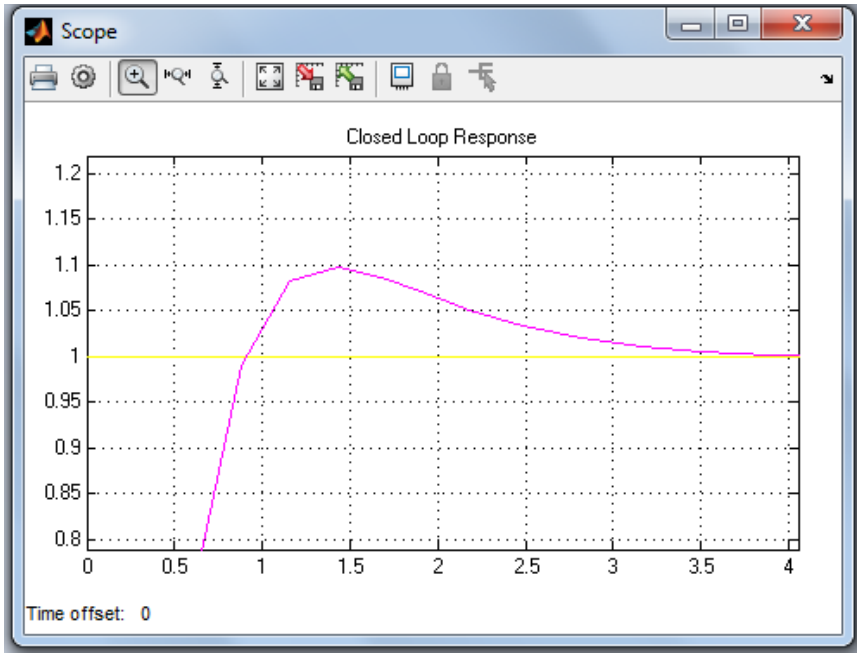


Figura 6.13 Salida optimizada ampliada del sistema de segundo orden críticamente amortiguado.

Ahora sí, podemos observar en profundidad los resultados experimentales (Ver Tabla 6.6) para comprobar si el sistema controlado cumple las especificaciones exigidas al principio del trabajo.

PARÁMETROS	SISTEMA CONTROLADO
Tiempo de establecimiento, Ts	2.2 segundos
Tiempo de sobreoscilaión, Tp	1.5 segundos
Sobreoscilaión, Mp (%)	10%

Tabla 6.6: Solución del sistema de segundo orden críticamente amortiguado.

Atendiendo a estos resultados, comprobamos que el tiempo de estabilización es menos de la mitad de los 5 segundos que nos propusimos en las especificaciones iniciales, por lo que consideramos que esta respuesta es muy satisfactoria. Por otro lado, podemos observar como la sobreoscilación máxima se da a los 1.5 segundos del arranque y alcanza un valor del 10% respecto a la referencia. Estos valores son inferiores a los dados en las especificaciones máximas que nos propusimos.

Por todo esto, concluimos que el sistema de control se ha optimizado de manera satisfactoria ante los requisitos impuestos.

6.2.3 Sistema sobreamortiguado

Tras la optimización, mediante *Differential Evolution*, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de segundo orden sobreamortiguado, Matlab nos devuelve las estimaciones de la Tabla 6.7:

Constante proporcional (K1)	0.6331
Constante derivativa (K2)	0.0461
Constante de ganancia (K3)	203.2252

Tabla 6.7: Constantes parametrizadas del sistema de segundo orden sobreamortiguado.

En este último sistema, comprobamos como la constante proporcional continua siendo de orden inferior a la unidad por el diseño utilizado. Esta vez, el control derivativo resulta ser muy pequeño, lo que nos da una idea de gran estabilidad reduciendo el tiempo de respuesta transitoria en gran medida una vez que el sistema sea optimizado. El valor de la ganancia continua siendo elevado como en todos los casos anteriores para corregir el error que se esté cometiendo.

El resultado visual de nuestro experimento queda ilustrado a continuación en la Figura 6.14:

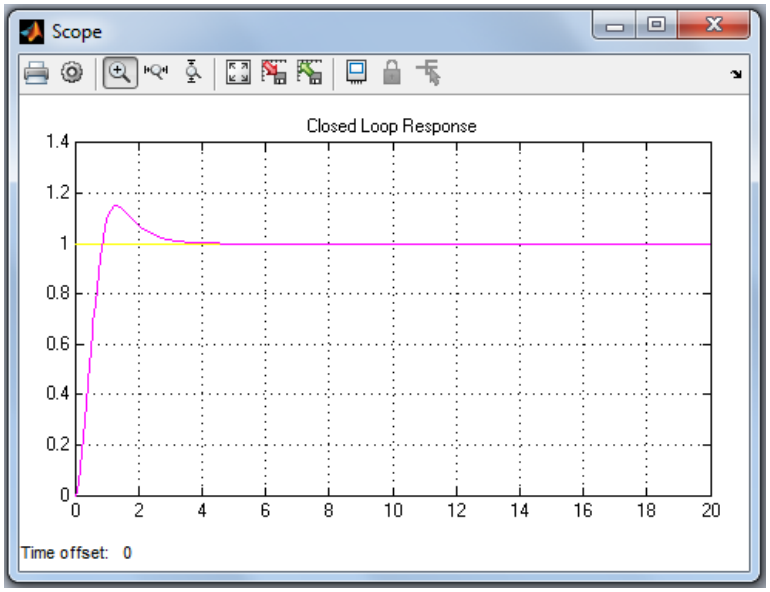


Figura 6.14: Salida optimizada del sistema de segundo orden sobreamortiguado.

Para analizar mejor los resultados, ampliaremos la zona límite entre la respuesta transitoria y la zona de régimen permanente en la Figura 6.15:

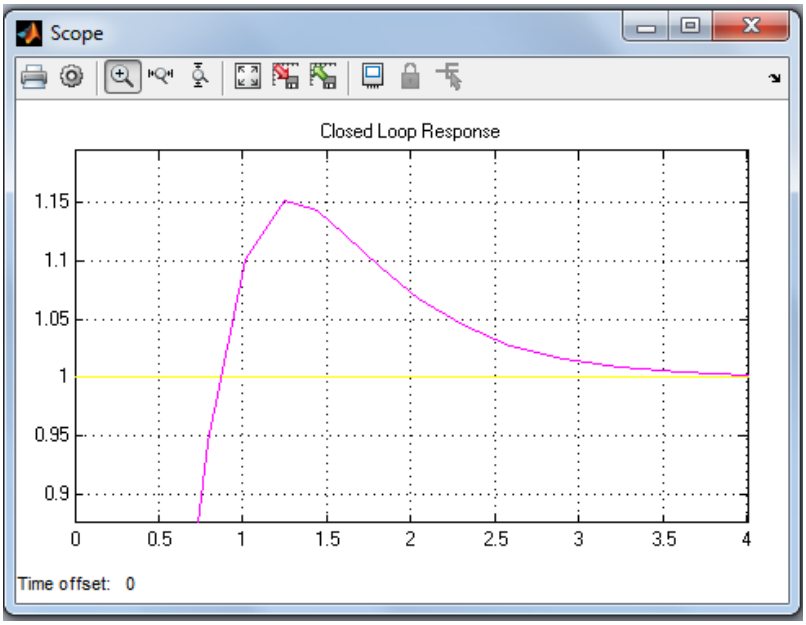


Figura 6.15: Salida optimizada ampliada del sistema de segundo orden sobreamortiguado.

Ahora sí podemos observar en profundidad los resultados experimentales (Ver Tabla 6.8) para comprobar si el sistema controlado cumple las especificaciones exigidas al principio del trabajo.

PARÁMETROS	SISTEMA CONTROLADO
Tiempo de establecimiento, Ts	2.25 segundos
Tiempo de sobreoscilación, Tp	1.25 segundos
Sobreoscilación, Mp (%)	15%

Tabla 6.8: Solución del sistema de segundo orden sobreamortiguado.

Según estos resultados, comprobamos que, al igual que en el sistema críticamente amortiguado, en el sistema sobreamortiguado el tiempo de estabilización es menos de la mitad de los 5 segundos que nos propusimos en las especificaciones iniciales, por lo que volvemos a considerar esta respuesta como muy satisfactoria.

La sobreoscilación máxima se da a los 1.25 segundos del arranque y alcanza un valor del 15% respecto a la referencia. En comparación con el sistema críticamente amortiguado, en el sistema sobreamortiguado conseguimos menor tiempo de sobreoscilación pero en contra puesta el porcentaje de sobreoscilación es mayor.

Finalmente, concluimos que el sistema de control se ha optimizado de manera satisfactoria ante los requisitos impuestos.

6.2.4 Sistema con polo en el origen del lugar de las raíces

Tras la optimización, mediante *Differential Evolution*, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de segundo orden con un polo en el origen, el controlador no es capaz de regular el sistema. En la Figura 6.16, se muestra la mala salida que obtuvimos:

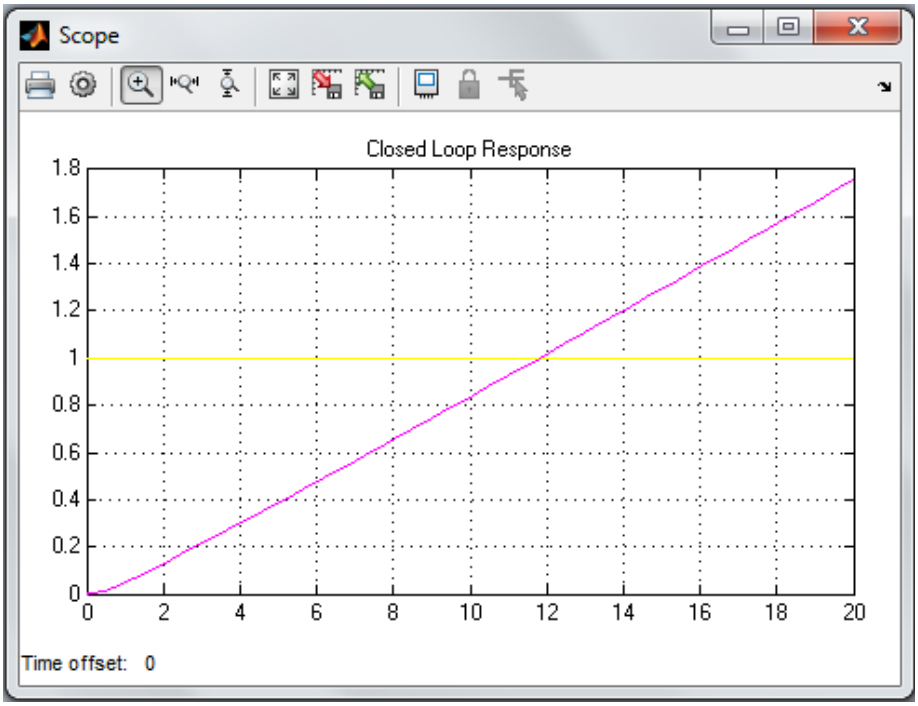


Figura 6.16: Salida optimizada del sistema de segundo orden con polo en el origen.

En un intento por conseguir controlar el sistema, tal y como comentamos en el apartado 3, implementación de *Differential Evolution*, incorporaremos paralelamente al controlador *fuzzy*, un regulador integrador PI. Debido a que nuestro algoritmo de optimización no es capaz de generarnos una constante de integración adecuada, deberemos estimarla manualmente mediante ensayo y error hasta lograr una salida favorable para su estudio. En la Figura 6.17 se muestra como queda el circuito con la anexión del regulador PI:

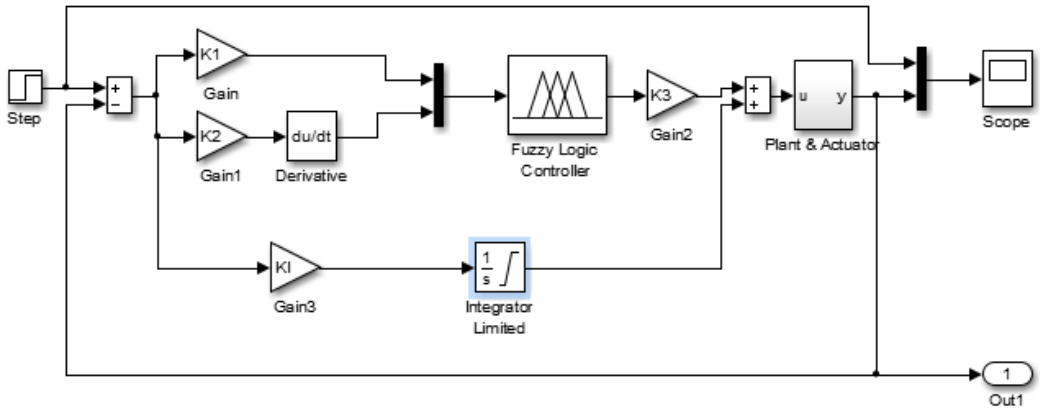


Figura.17: Circuito de simulación del control de sistemas con PI en paralelo.

A continuación, mostraremos resumido el progreso de ensayos realizados en busca de calibrar la constante de integración del regulador PI. Vea Figuras 6.18-6.23. Al lado, podrá observar el valor de los parámetros optimizados por el algoritmo (K1, K2, y K3) (Ver Tablas 6.9-6.13), junto con el valor de la constante de integración con la que se llevó a cabo el ensayo.

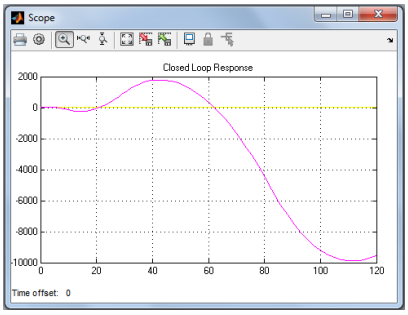
<table>
<tr><th>CONSTANTE DE INTEGRACIÓN</th><th>PARÁMETROS OPTIMIZADOS</th><th>SALIDA OBTENIDA</th></tr>
<tr><td>Ki = 10</td><td>
<table>
<tr><td>Constante proporcional (K1)</td><td>0.1568</td></tr>
<tr><td>Constante derivativa (K2)</td><td>0.0468</td></tr>
<tr><td>Constante de ganancia (K3)</td><td>0.9873</td></tr>
</table>
Tabla 6.9: Constantes parametrizadas del sistema con polo en el origen usando Ki=10.
</td><td>


Figura 6.18: Salida optimizada del sistema de segundo orden con polo en el origen usando Ki=10.
</td></tr>
</table>

Ki = 1

Constante proporcional (K1)	0.7849
Constante derivativa (K2)	0.0356
Constante de ganancia (K3)	0.1587

Tabla 6.10: Constantes parametrizadas del sistema con polo en el origen usando Ki=1.

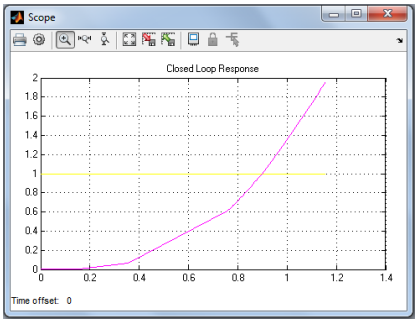


Figura 6.19: Salida optimizada del sistema de segundo orden con polo en el origen usando Ki=1.

Ki = 0.5

Constante proporcional (K1)	0.7653
Constante derivativa (K2)	0.0356
Constante de ganancia (K3)	0.2676

Tabla 6.11: Constantes parametrizadas del sistema con polo en el origen usando Ki=0'5.

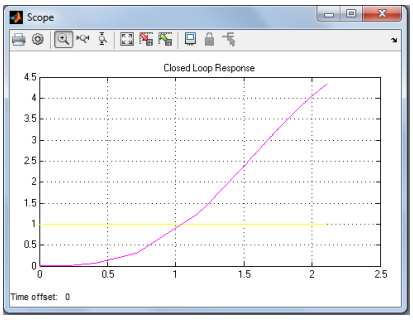


Figura 6.20: Salida optimizada del sistema de segundo orden con polo en el origen usando Ki=0'5.

Ki = 0.1

Constante proporcional (K1)	0.4682
Constante derivativa (K2)	0.0469
Constante de ganancia (K3)	0.6348

Tabla 6.12: Constantes parametrizadas del sistema con polo en el origen usando Ki=0'1.

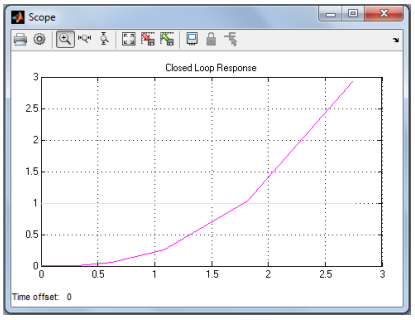


Figura 6.21: Salida optimizada del sistema de segundo orden con polo en el origen usando Ki=0'1.

Ki = 0.01

Constante proporcional (K1)	0.7963
Constante derivativa (K2)	0.0689
Constante de ganancia (K3)	0.4685

Tabla 6.13: Constantes parametrizadas del sistema con polo en el origen usando Ki=0'01.

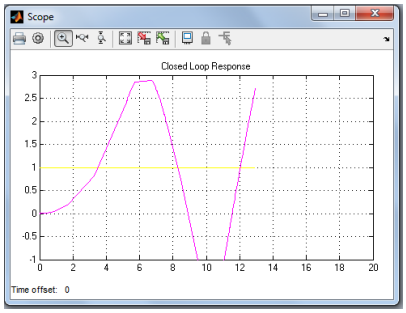


Figura 6.22: Salida optimizada del sistema de segundo orden con polo en el origen usando Ki=0'01

Finalmente utilizando **'KI = 0.001'** obtenemos las constantes parametrizadas de la Tabla 6.14:

Constante proporcional (K1)	0.2142
Constante derivativa (K2)	0.1678
Constante de ganancia (K3)	0.6280

Tabla 6.14: Constantes parametrizadas del sistema con polo en el origen usando Ki=0'001.

En la Figura 6.23 comprobamos como el sistema resulta comportarse como oscilador. Concluimos que nuestro regulador *fuzzy* no es capaz de controlar los desequilibrios ocasionados por polos en el origen del lugar de las raíces, a pesar de contar con la ayuda de un regulador PI para corregir la acción integral. Resulta llamativo destacar que para conseguir mínimamente un control, el algoritmo nos responde con una ganancia a la salida del controlador comprendida entre 0 y 1, dándonos a entender que trata de entregar señales muy débiles a la entrada del sistema y de este modo se minimice la inestabilidad a pesar de no poder hacerla desaparecer.

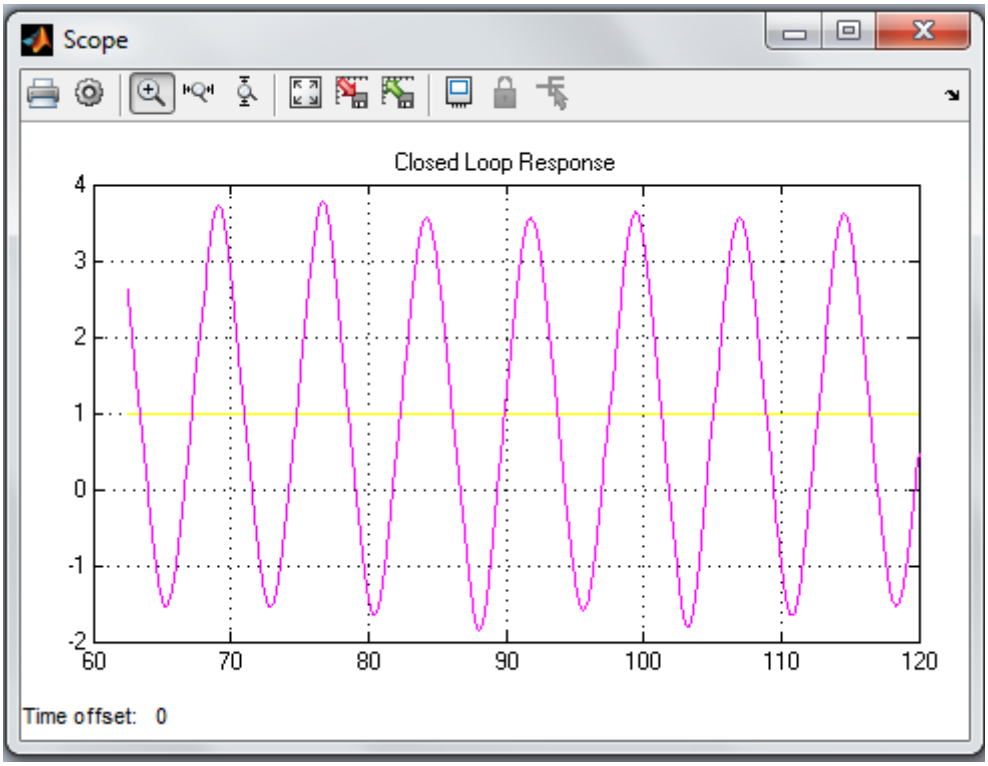


Figura 6.23: Salida optimizada del sistema de segundo orden con polo en el origen. usando Ki=0'001.

6.2.5 Sistema con polo inestable

Tras la optimización, mediante *Differential Evolution*, de las constantes parametrizadas (K1, K2, y K3) que regulan el bloque *fuzzy* para el control del sistema de segundo orden con un polo inestable, el algoritmo no es capaz de regular el sistema. Tras múltiples intentos de ensayos sobre el sistema, terminamos concluyendo que somos incapaces de controlarlo. Sabemos que añadir el regulador PI en paralelo al *fuzzy* tampoco ayudaría a controlar la salida de este sistema. Este resultado era el esperado por contar con el polo en situación de inestabilidad. En la Figura 6.24, apreciamos como la salida queda inestable desde el arranque del sistema, y podemos observar una tendencia a infinito como respuesta de salida.

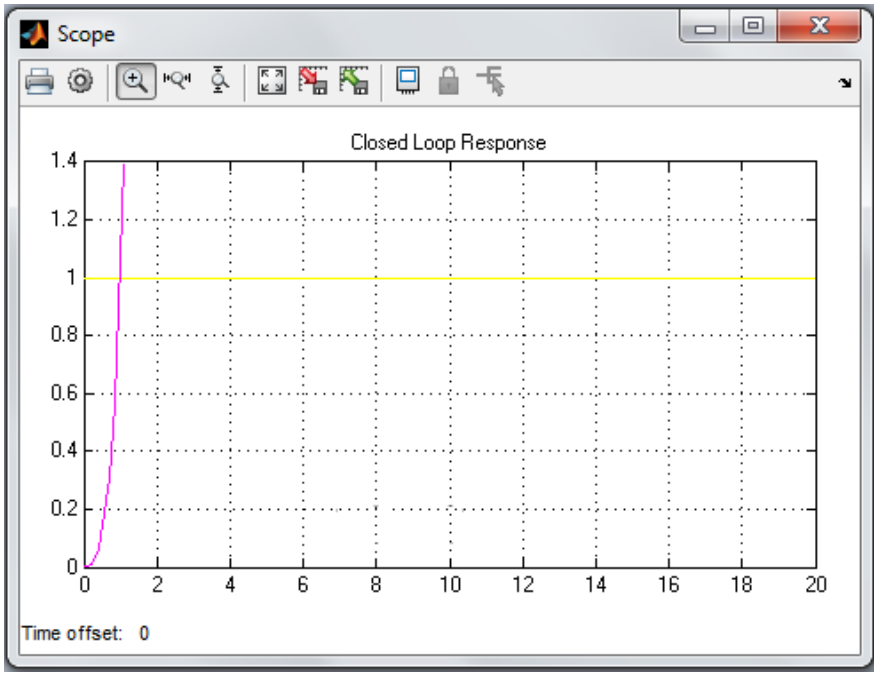


Figura 6.24: Salida optimizada del sistema de segundo orden con polo inestable.

(Véase referencias bibliográficas [1], [2], [5] y [6]).

7 Conclusión y trabajo futuro

El universo es una colección de objetos que se rigen por las leyes naturales y por la lógica. Un conjunto en el universo es una colección de objetos, de tal manera que es posible decir si un objeto cualquiera pertenece o no a ese conjunto. Esto lo logramos gracias a la lógica específica, en la que podemos utilizar el '1' y el '0' (verdadero y falso) para determinar la pertenencia o la no pertenencia en función de si el objeto cumple una determinada propiedad inherente al conjunto.

Sin embargo, a lo largo del trabajo hemos comentado que el ser humano es capaz de utilizar predicados difusos que crean imprecisiones. La teoría de conjuntos difusos permite la extensión de grados de pertenencia (nulo, parcial o total) a conjuntos que no tengan bien definidas sus fronteras y cuenten con funciones que admitan la pertenencia parcial utilizando valores comprendidos en el intervalo [0, 1].

Después de casi 30 años investigando la lógica difusa, se ha podido observar su gran potencial, y los ingenieros se han aprovechado de esto para modelar y controlar sistemas no lineales, dinámicamente complejos o que cuenten con combinaciones de entrada y salida inusuales. Aun así, es mejor evitar su uso en los casos en que el control convencional (PID) rindiera con resultados satisfactorios o cuando el modelo matemático sea fácilmente soluble, ya que como dicen todos los grandes maestros de la ingeniería: *"Si algo funciona bien, no lo toques"*. Debemos dejar claro que los controladores borrosos han sido desarrollados para acciones de control que queden fuera del alcance de los controladores convencionales, y no para ser sustitutos de estos.

El dominio computacional de la lógica difusa, que permite procesar valores parciales de veracidad en conjuntos, ha sido de gran ayuda para que la ingeniería logre hacer avances en muchos ámbitos de la ciencia. Entre otros, podemos destacar algunos de estos ámbitos en los que hoy día se están investigando:

- Diagnósticos médicos.
- Control de sistemas en tiempo real: control de tráfico, control de aeropuertos, etc.
- Predicción climatológica.
- Control de maquinaria pesada: motores de ascensores, compuertas…
- Simulaciones de realidad virtual.

Es muy importante señalar que los controladores basados en lógica difusa requieren una excelente depuración y prueba antes de ser comerciales. Actualmente no existe ningún análisis matemático que garantice la total estabilidad. Por otro lado, es difícil crear una regla confiable sin la participación de una persona experta acostumbrada a realizar manualmente el trabajo que deseamos que desempeñe el sistema. Este operario resulta tan imprescindible como el propio ingeniero a la hora de diseñar el controlador.

Aun así, gracias al formato de las reglas obtenemos controladores simples y robustos capaces de responder ante cambios en el sistema, y obtener una mayor tolerancia ante ruidos que perturben la estabilidad.

Por otro lado, también debemos destacar la importancia de la aparición de potentes computadores capaces de procesar gran cantidad de información, permitiendo resolver problemas de gran complicación ingenieril que antes eran imposibles. Hoy en día, los algoritmos genéticos siguen alcanzando grandes éxitos de optimización en ingeniería como pudiesen ser: diseño de circuitos, estrategias de mercado, acústica, astronomía, física, química, etc.

La gran ventaja con la que cuenta los algoritmos evolutivos es que actúan intrínsecamente paralelos, es decir, son capaces de explorar el espacio de soluciones utilizando descendencias múltiples, permitiendo hacer varias evaluaciones al mismo tiempo. Resulta imprescindible analizar los resultados mediante una función de costes, la cual siempre devuelve un valor real positivo que depende de la satisfacción con que se cumplen las especificaciones del problema. A medida que se suceden las iteraciones hasta lograr la solución global, este valor tiende a descender. De no ser así, se debe concluir que el algoritmo no es capaz de resolver con satisfacción el problema. Ello puede ser debido a que estemos empleando parámetros mal calibrados en la optimización, o que no exista físicamente un desenlace satisfactorio. En la Tabla 7.1 podemos observar la evolución de los valores devueltos por la función de costes, *bestval*, a lo largo de las 40 iteraciones en el sistema de primer orden y los sistemas de segundo orden subamortiguado, críticamente amortiguado y sobreamortiguado. Para visualizar mejor la tendencia descendiente de este valor, acompañaremos unas gráficas en las Figuras 7.1-7.4:

Iteración	Sistema primer orden	Sistema subamortiguado	Sistema críticamente amortiguado	Sistema sobreamortiguado
10	1,049504	1,904166	1,188101	1,451848
20	1,049504	1,776131	1,113158	1,294348
30	1,048295	1,300222	1,100073	1,288287
40	1,044866	1,242697182	1,08984	1,280078

Tabla7.1: Evolución del valor 'bestval' devuelto por la función de costes a lo largo de las iteraciones.

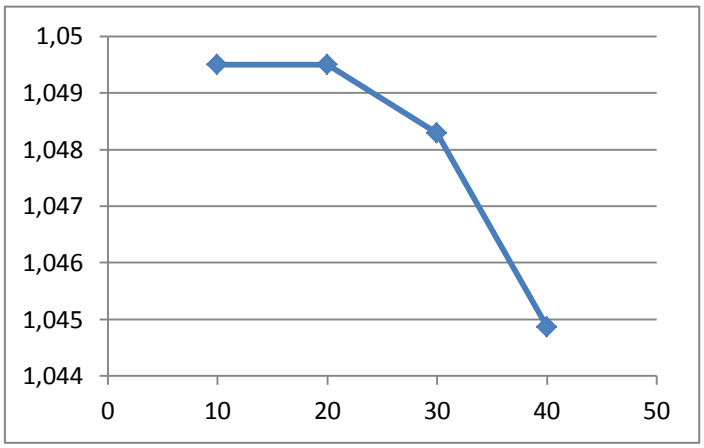


Figura 7.1: Evolución de la función de costes en sistema de primer orden.

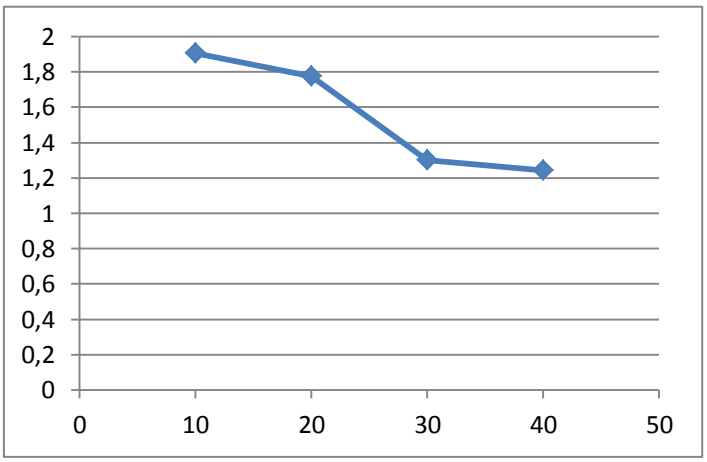


Figura 7.2: Evolución de la función de costes en sistema subamortiguado.

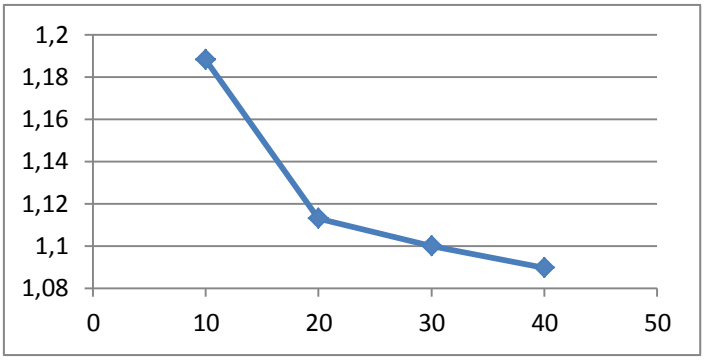


Figura 7.3: Evolución de la función de costes en sistema críticamente amortiguado.

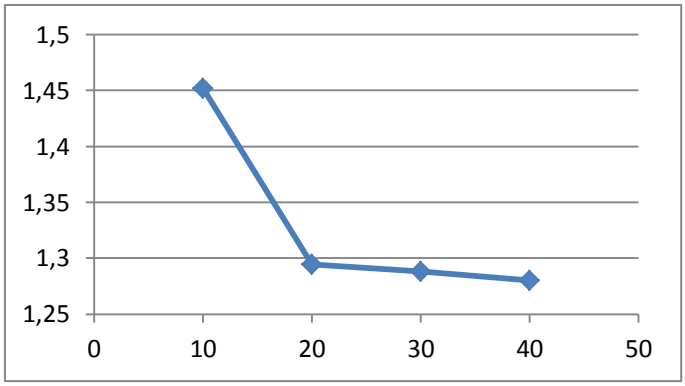


Figura 7.4: Evolución de la función de costes en sistema sobreamortiguado.

Gracias a esta tendencia descendente que podemos observar en las Figuras 7.1-7.4, concluimos que *Differential Evolution* es un algoritmo perfectamente indicado en la optimización de los parámetros del controlador *fuzzy* en el control de sistemas de primer y segundo orden que posean estabilidad.

En cuanto al sistemas con polo en el origen y al sistema con polo inestable, ultimamos que el *Differential Evolution* no es apto en la optimización, tal y como esperábamos al estudiarlos. El algoritmo no es capaz de evolucionar y concluye las iteraciones al darse cuenta de ello, por lo que no devuelve ningún *bestval*.

Aunque los algoritmos evolutivos ya han demostrado su eficiencia y potencial, sabemos que tienen ciertas limitaciones, sin embargo, se estudia como poder superarlas ya que ha quedado demostrada la validez de la evolución genética. Se estudia desarrollar lenguajes robustos capaces de

producir resultados sin errores en la mutación. Otro problema conocido es la convergencia prematura, es decir, que el algoritmo merme la diversidad demasiado pronto y provoque una optimización local en vez de una global. Para evitar este problema es necesario realizar una buena selección de los parámetros de optimización antes de empezar a trabajar.

Finalmente, se aconseja no utilizar los algoritmos evolutivos en problemas resolubles de manera analítica. Esto no se debe a que resulten incapaces de encontrar soluciones, sino que consumen tiempo y potencia innecesarios, sabiendo que matemáticamente llegan a la misma solución.

Meses antes en un trabajo anterior, en la asignatura de control inteligente, realizamos la calibración de los parámetros de un regulador *fuzzy* para el control de sistemas (contantes proporcional, derivativa y la ganancia del regulador).

Aquella vez no empleamos *Differential Evolution* en la optimización de las constantes, sino que la calibración fue realizada de manera heurística, estudiando la salida con que respondían los sistemas, al igual que hemos actuado en la optimización del sistema que contaba con un polo en el origen al calibrar la constante de integración '*Ki*'. Fueron numerosos los ensayos que tuvimos que realizar hasta lograr una salida que cumpla los requisitos requeridos, puesto que todas las variaciones que realizábamos en los parámetros provocaban una salida distinta a la anterior, y ésta podía ser mejorada o empeorada. Ya que no contábamos con una función de coste, era nuestra interpretación la que usábamos para decidir si estábamos en el buen camino hacia la optimización de los parámetros.

Gracias a la realización del presente TFG, he podido experimentar la gran ventaja que ha supuesto poder utilizar algoritmos evolutivos para hallar los valores que sean capaces de optimizar los reguladores de control. Esta ventaja está clarificada en el tiempo que se logra ahorrar al emplear *Differential Evolution*. Además, al contar con un algoritmo bien programado, que a su vez cuente con una función de costes centrada en la minimización del tiempo de estabilización (Ts), sobreoscilación máxima (Mp) y tiempo de sobreoscilación máxima, podemos asegurar que los resultados obtenidos son los mejores optimizados a nivel global dentro del amplio rango de soluciones. *(Véase referencias bibliográficas [11], [13], [15] y [20])*.

8 Anexos

8.1 Planificación del proyecto

Este Trabajo Fin de Grado se ha realizado desde Diciembre de 2013 hasta Julio de 2014. De manera general, se han seguido las siguientes etapas a lo largo del mismo:

1. Estudio del regulador *fuzzy* y del algoritmo de *Differential Evolution*
2. Análisis de sistemas mediante función de transferencia
3. Enfoque práctico: Implementación experimental y simulaciones
4. Análisis y conclusiones de los resultados

La lista de tareas, así como su comienzo, duración y final queda desgranada en la Tabla 8.1, que acto seguido da lugar al Diagrama de Gantt representado en la Figura 8.1.

Tarea	Fecha de inicio	Duración (días)	Fecha de finalización
Estudio del regulador fuzzy y del algoritmo de Differential Evolution	20/12/2013	10	30/12/2013
Análisis de sistemas mediante función de transferencia	10/01/2014	20	20/01/2014
Enfoque práctico: Implementación experimental y simulaciones	21/01/2014	70	30/03/2014
Análisis y conclusiones de los resultados	15/06/2013	15	30/06/2014
Realización del informe final	01/07/2014	15	31/07/2014

Tabla 8.1: Planificación por tareas del Trabajo Fin de Grado

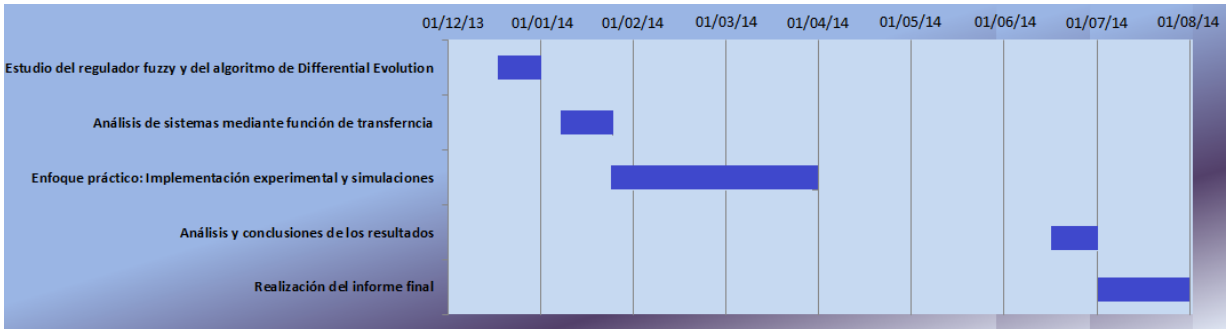


Figura 8.1. Diagrama de Gantt del Trabajo Fin de Grado.

8.2 Presupuesto del proyecto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
José Antonio Hernaiz Navas

2.- Departamento:
Sistemas y Automática

3.- Descripción del Proyecto:

- Titulo	OPTIMIZACION DE UN CONTROLADOR FUZZY MEDIANTE DIFFERENTIAL EVOLUTION
- Duración (meses)	5
Tasa de costes Indirectos:	10%

4.- Presupuesto total del Proyecto (valores en Euros):
8.539 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)
					0,00
Garrido Bullón, Luis Santiago		Development Supervisor	0,31	4.289,54	1.329,76
Hernaiz Navas, José Antonio		Ingeniero	2,3	2.694,39	6.197,10
					0,00
					0,00
		Hombres mes	**2,61**	**Total**	7.526,85

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador de desarrollo	2.000,00	100	5,0	60	166,67
				Total	166,67

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Matlab&Simulink Stuent Suite	MathWorks	69,00
	Total	69,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	7.527
Amortización	167
Costes de funcionamiento	69
Costes Indirectos	776
Total	8.539

9 Índice de figuras

10 Índice de tablas

11 Referencias bibliográficas

[1] Ogata, K. (2010), *Modern Control Engineerin.*5ª ed. Prentice Hall, New Jersey, and ISBN: 10: 0-13-615673-8

[2] Dorft, R.; Bishop, R. H., (2005), *Moderm Control Systems.* 15ª ed., Pearson Educación S. A., Madrid, and ISBN: 84-205-4401-9

[3] Magrab, E. B. et al. (2011), *An engineer's guide to MATLAB with applications from Mechanical, Electrical, Civil and Biology Systems Engineering.* 3ª ed., Prentice Hall, New Jersey. ISBN: 10: 0-13-703954-9

[4] López Pérez, C. (2002), *Matlab y sus aplicaciones en las ciencias y la ingeniería,* Prentice Hall, Madrid.

[5] Matía, F.; Jiménez, A. (2003), *Teoría de sistemas.* Sección de publicaciones de la Escuela Técnica Superior de Ingenieros Industriales. Universidad Politécnica de Madrid

[6] Oppenheim, A. V.; Willsky, A. S.; Nawab, S. H. (1998), *Señales y Sistemas.* 2ª ed., Prentice Hall, Madrid, and ISBN: 9789701701164

[7] Trillas, E. (1992), *Fundamentos e introducción a la ingeniería Fuzzy,* Omron Electronics, S. A., Madrid.

[8] Trillas, E.; Gutierrez, J. (1994), *Aplicaciones de la Lógica Borrosa,* CSIC, Raycar S. A., Madrid and ISBN: 84-00-07271-5

[9] Trillas, E.; Alsina, C.; Terricabras, J. M. (1995), *Introducción a la Lógica Borrosa,* Ariel, Barcelona.

[10] Legerén, J.; Núñez, A.; Ortiz, J. (2012), *Introducción a los algoritmos genéticos,* Universidad Politécnica de Madrid, Madrid.

[11] Marczyk, A. *Algoritmos genéticos y computación evolutiva.* (Online). Último acceso: 04/02/2014. Disponible en: <http://the-geek.org/docs/algen/>

[12] Razón Artificial. *Algoritmos genéticos.* (Online). Último acceso: 04/02/2014. Disponible en: <http://razonartificial.com/2010/09/algoritmos-geneticos/>

[13] Dominguez-Dorado, M. "Programación de algoritmos genéticos". *Todo Linux.* Nº 12 (2005). Pág. 16-20.

[14] Price, K. P.; Storn, R. M.; Lampinen, J. A. (2005), *Differential Evolution: A practical approach to global optimization,* Springer, New York, and ISBN-10 -3-540-20950-6

[15] Feoktistov, V. (2006), *Differential Evolution: In search of solutions,* Springer, New York, and ISBN-10:0-387-36895-7

[16] Kenneth Price; Rainer Storn. *Differential Evolution for continuous funtion optimization.* (Online). Último acceso: 06/03/2014. Disponible en: <http://www1.icsi.berkeley.edu/~storn/code.html>

[17] Colaboradores de Wikipedia, *Evolucion diferencial* (Online). Wikipedia, la enciclopedia libre. Último acceso: 06/03/2013. Disponible en: <http://es.wikipedia.org/wiki/Evoluci%C3%B3n_diferencial>

[18] Merelo, J. J., *Informatica evolutiva: Algoritmos genéticos* (Online). Último acceso: 25/02/2014. Disponible en: <http://geneura.ugr.es/~jmerelo/ie/ags.htm>

[19] Gestal, M., *Introducción a los algoritmos genéticos* (Online). Depto. Tecnologías de la Información y las Comunicaciones. Universidade da Coruña. Último acceso: 25/02/2014. Disponible en: <http://sabia.tic.udc.es/mgestal/cv/aaggtutorial/aagg.html>

[20] Recio, J. A. "Introducción a la lógica borrosa". *Hiperenciclopedia de divulgación del saber.* Vol.8, Nº 3 (2014). Pág. 16-20.

[21] SAGO OPTIMIZATION, *Evolución diferencial con la competencia de control de parámetros de ajuste* (Online). Último acceso: 08/03/2013. Disponible en: <http://www1.osu.cz/home/habibal/optimization/differential_evolution.html#id20>

[22] Colaboradores de Wikipedia, *Ingeniería de control* (Online). Wikipedia, la enciclopedia libre. Último acceso: 06/05/2013. Disponible en: <http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_control>

[23] Olmo, M. A., *Tutorial de introducción de lógica borrosa* (Online). Último acceso: 06/02/2013. Disponible en: <http://www.dma.fi.upm.es/java/fuzzy/tutfuzzy/indice.html>